“Everything Personal, Not Just Business”: Improving User Experience Through Rule-Based Service Customization

Richard Hull, Bharat Kumar, Daniel Lieuwen,
Peter F. Patel-Schneider, Arnaud Sahuguet,
Sriram Varadarajan, and Avinash Vyas

Network Data and Services Research
Bell Labs, Murray Hill
http://db.bell-labs.com

December 16, 2003

The Challenge

- Services are becoming more varied and more complex
  - Web services paradigm
    - SOAP, WSDL, BPEL, …
  - Converged services – telecom + web
    - Parlay/OSA, SIP, 3GPP GUP, …
- Key to mass deployment is ***support for personalization***
  - "Enter Once, Share Everywhere" of profile data
    - See [Sahuguet et. al., CIDR 2003]
  - Enable capture/execution on personal preferences
    - Focus of this talk
- Converged services provide concrete examples ***today***, e.g.,
  - Location-based services: privacy of my location
  - Selective Reach Me (SRM): "intelligent" call forwarding
  - Another example in paper: services selection

How to support context-aware, preference-driven decisions
- High speed
- Easy to maintain
- Appropriate for mass deployment (generic, scalable, cheap)

Outline

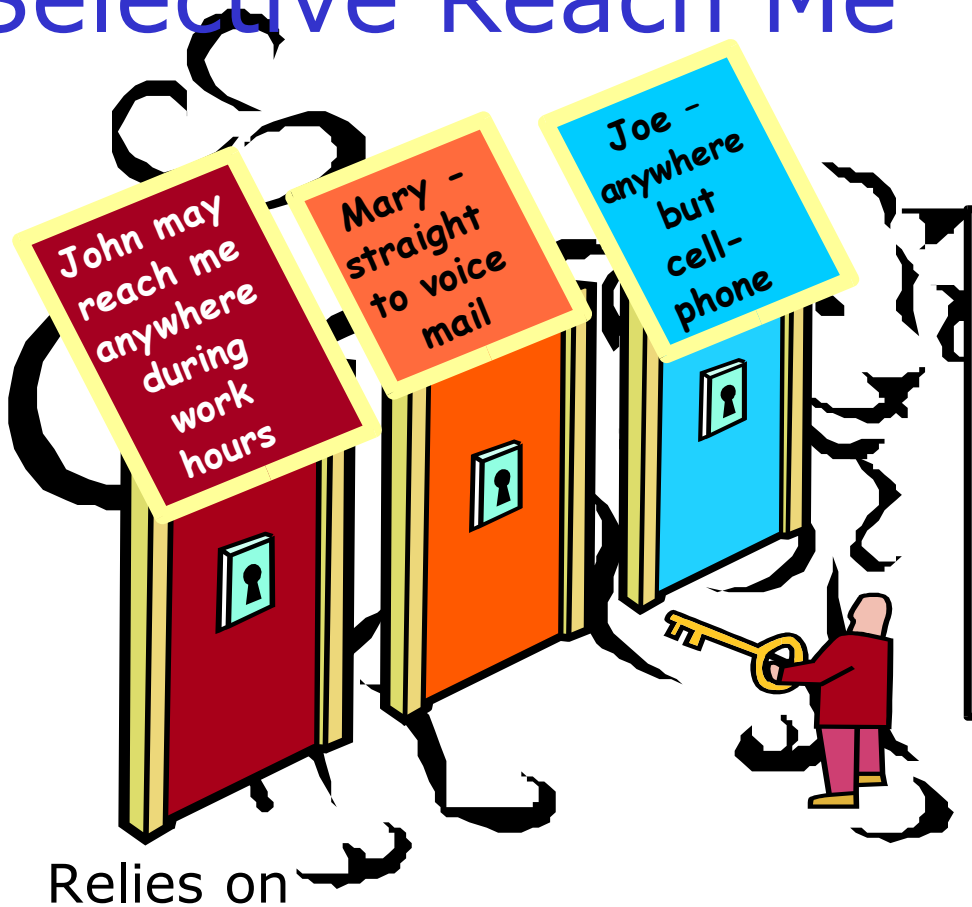
- Motivating examples
- The Houdini approach
- The Houdini rules engine
- Self-provisioning framework
- Related work

iLocator: Location-based Track & Alert

(developed by Vishy Poosala's group in Bell Labs)

- Allows tracking of people/events/enterprises and displays their positions on a real-time map (or sends alerts when they are in the user's vicinity)
- People: Like Instant Messenger, maintains a buddy list and automatically informs about buddies entering or leaving a defined radius around the current user location
  - E.g., let me know when a friend is within 50 km from me
  - Or, when my kid moves more than 10 km away
- Key areas for personalization
  - Who/what to display on my screen
  - Who can see my location under what circumstances

Selective Reach Me



"Call people, not devices"

Enable selected users to contact me in realtime, according to my preferences, and without knowing in advance what device to reach me with; block other callers

- Relies on
  - Context: presence info, caller, recent/current activity, calendar, …
  - Preferences: privacy, priorities of activities/callers, types of connection (circuit, cell, VoIP, SMS…), …
- Need to combine and reason about preferences
  - Is this "working hours"? How "important" is the caller? How "busy" am I? Do I "have time" to take this call now? …

Two Common Approaches to Personalization

Bell Labs

- Value-based policy enablement
  - Typical of on-line newspapers
  - Data structure is created to hold all supported preferences
    - This is interpreted at run time
  - Easy, intuitive for users; decisioning is usually transparent
  - ***Inflexible: adding new kind of preference ⇒ write new code***
- Automated learning using Bayesian nets
  - Typical of spam filters
  - System builds up Bayesian net according to user habits
  - Easy for users to "train" the system
  - ***Essentially impossible for users to understand or alter learned behaviors***

Houdini Approach to Personalizing Services

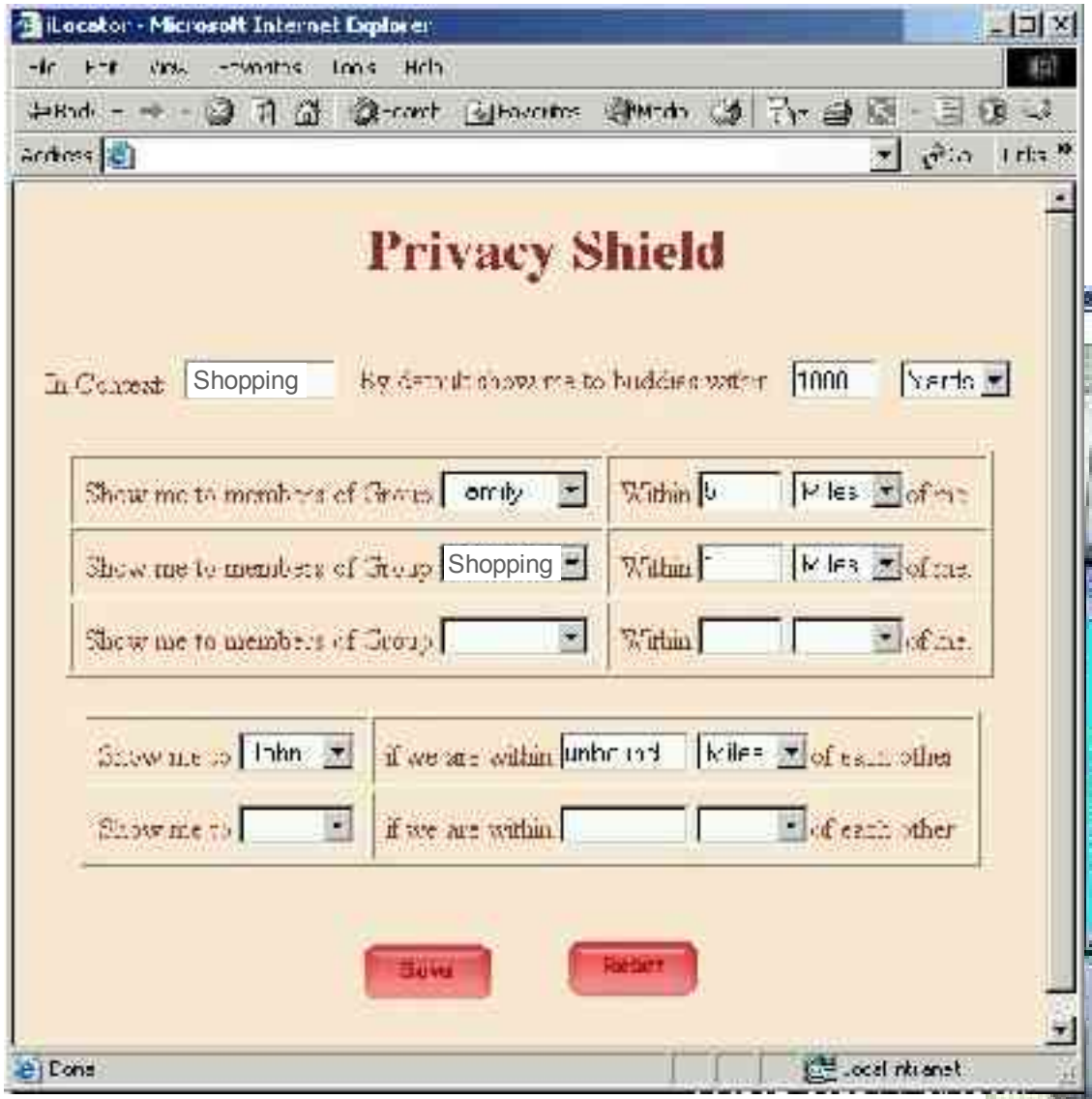
- Use rule-based policy enablement
  - Rules are high-level ⇒ cheaper to create/evolve rulesets
  - Rules are interpreted ⇒ can replace on the fly
  - Cheaper to support different applications
  - Cheaper to support different types of users for same application
- Trade-off of expressive power vs. performance
  - Houdini supports production style rules
  - Chaining but no cycles
- Provide framework for self-provisioning
  - End-users should fill web forms, not write rules

How are these pieces brought together?

Preferences for a Privacy Shield

- What people/applications should see my location, and under what circumstances?
- Example preferences
  - My boss can see my location whenever I'm "working"
  - If I'm in "shopping" context, then my "shopping buddies" and my "family" can see my location
  - Never show my location to telemarketer except Starbucks
- Various profile data needed to support privacy shield
  - Buddy lists, categories of buddies
  - Favorite (kinds of) stores, restaurants, events, …
  - Privacy desires based on different ***contexts*** ("work", "leisure", …)
- And, provisioning of preferences…

Provisioning Privacy Shield for given context (web or phone-based)



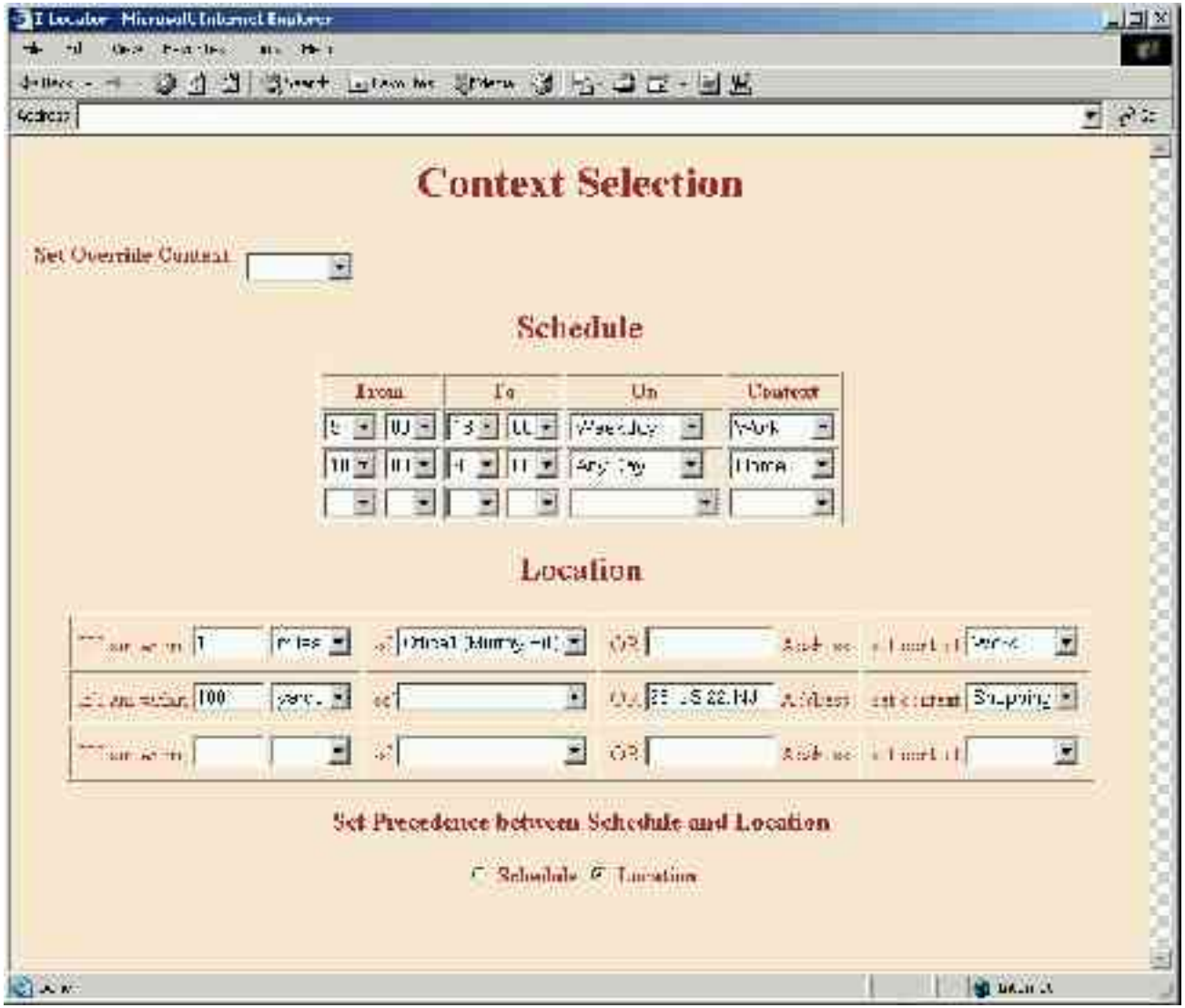
E.g., For each context, allow privacy shield based on distance to buddies:

- A default distance
  - E.g., all buddies within 1000 yards can see me
- A group-based distance
  - E.g., family members within 5 miles can see me
- An individual distance
  - E.g., John …

Explicitly setting context

- Subscribers can
  - View current context (which may be inferred)
  - Explicitly set context
- Issue: After a while, this will be cumbersome
  - Especially for the privacy shield, since subscriber's involvement with privacy shield is essentially passive
- Houdini framework permits end-user to specify preferences for inferring context

Provisioning preferences for inferring context

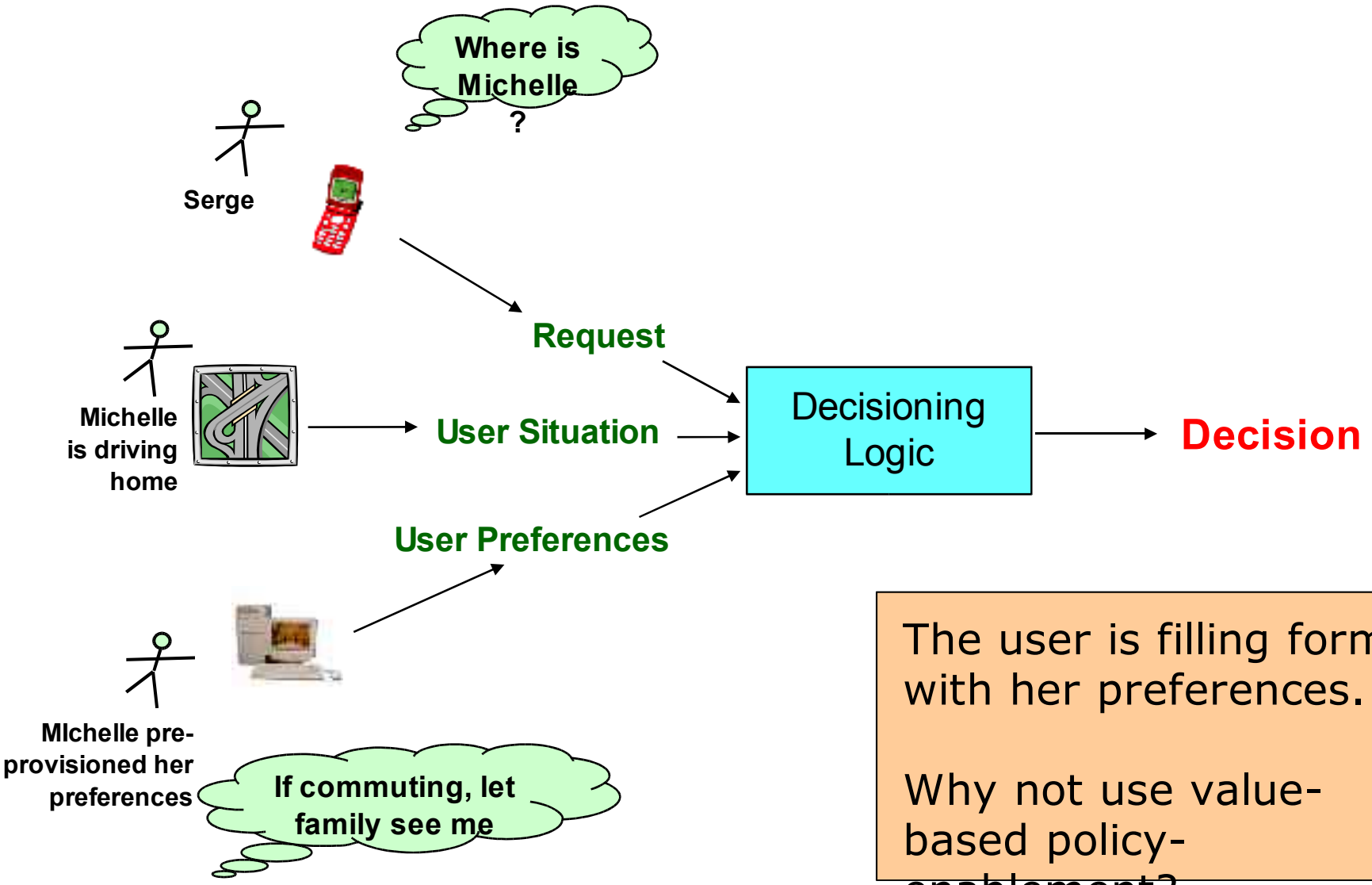


Context can be inferred if not specified explicitly, e.g.:

- if it's weekday between 9-6pm, use the Work context
- if user is within 1 mile of his office location, use the Work context

In principle, can include other data, e.g., calendar, recent device usage

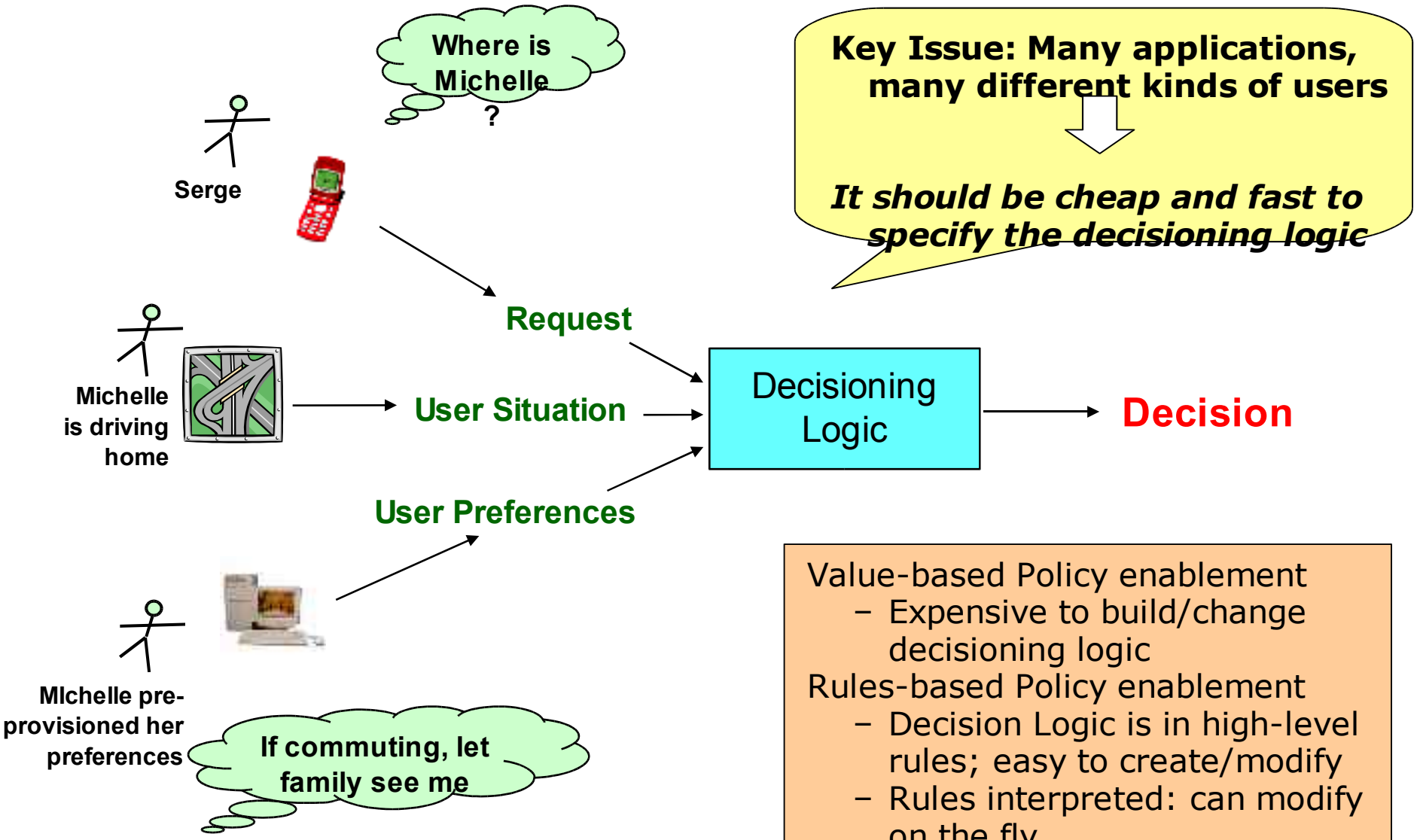
Schematic of Decisioning for Location Privacy



The user is filling forms with her preferences.

Why not use value-based policy-enablement?

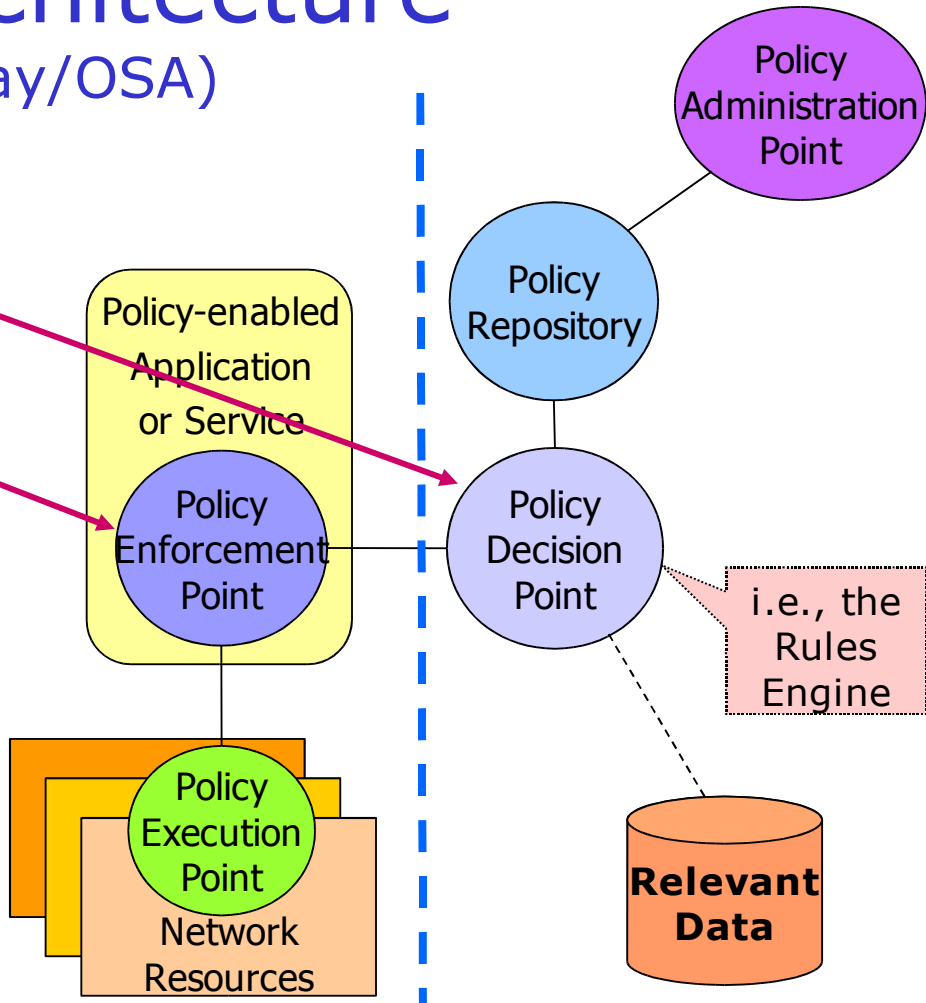
The case for rules-based policy-enablement



Policy Reference Architecture

(as found in, e.g., IETF and Parlay/OSA)

- ***Policy Decision Point***
  - the component making the decision
  - the decision in itself does not have any impact or side-effect
- ***Policy Enforcement Point***
  - the place in a component that enforces the decision
  - can be multiple Enforcement Points within a component
- Policy Execution Point
  - the place in a component actually performing the enforcement
  - (sometimes merged with Policy Enforcement Point)
- Policy Repository
  - the component storing the policies
- Policy Administration Point
  - for provisioning, checking policies



Separation of components forces
- Deliberate input/output perspective on decision requests
- Structured, more reliable use of policy engine

Key Elements of Houdini Policy Language
(Now in a Lucent product)

- Supports production-style rules, with chaining, e.g.,

```
If Monday 9 to 5 then Time_Category = "Working"
If Tuesday 10 to 3 then Time_Category = "Working"
...
        If Time_Category = "Working" and requester_category = "boss"
            then "reveal my location"
        If Time_Category = "Working" and requester_category = "friend"
            then "do not reveal my location"
...
```

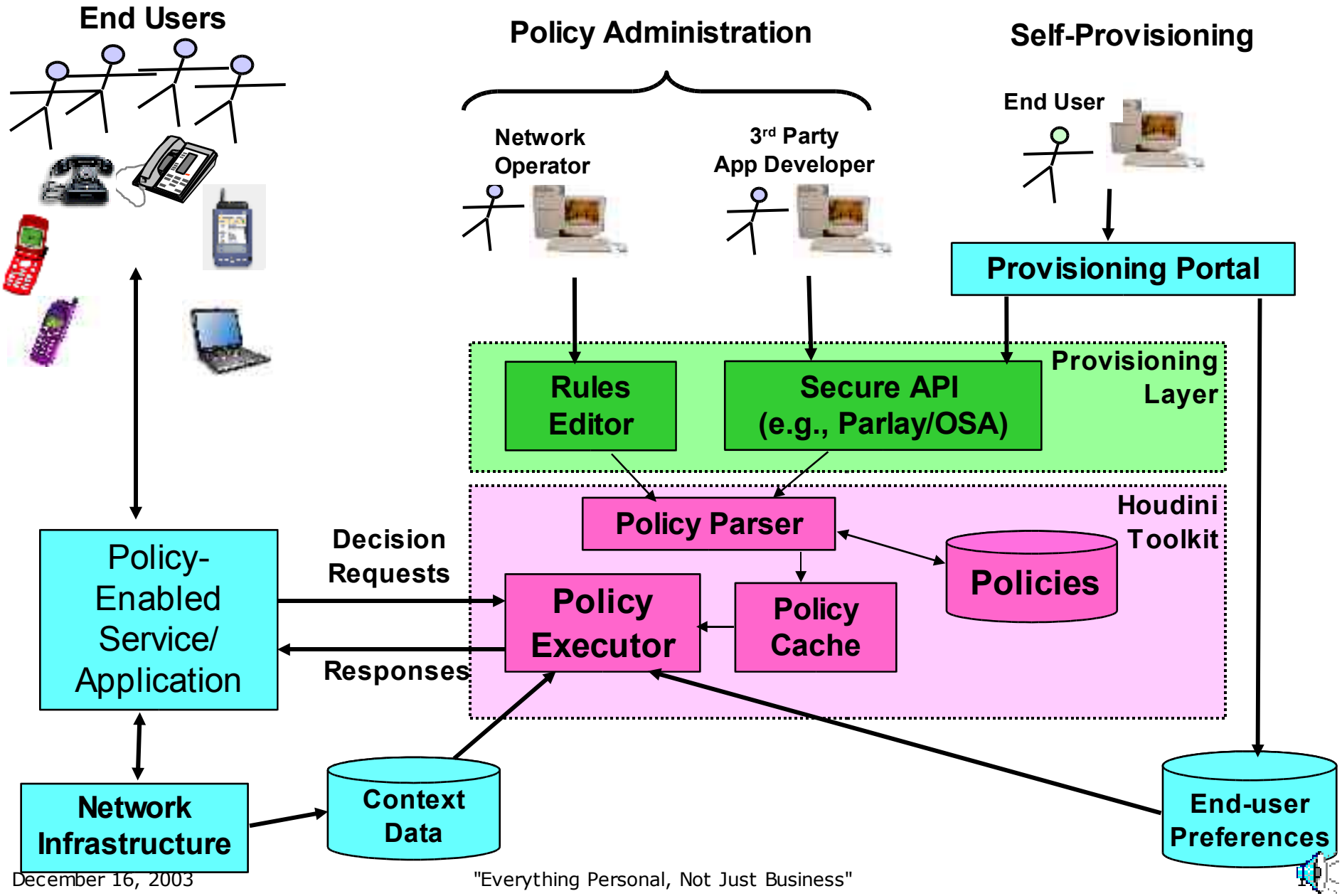
  - Rules with chaining are *more expressive than typical IETF rules*
  - Chaining supports *modularity* and *simplifies updates*
    - Use intermediate variables as "goals" for different modules
    - Single override rule can impact multiple rules, e.g., "I am working until 9PM today"
- Prohibits cycles/recursion in rules
  - Enables much *faster execution* times
- Data types: arbitrarily nested combination of atomic/record/list types
  - Strong typing of input, output and internal variables for rulesets
    - Permits *static type checking* and guarantees about run-time behavior
  - Can support filtering of lists, white-listing, black-listing
- "Support" functions extend the functionality of the rules engine

Houdini Rules Engine Performance

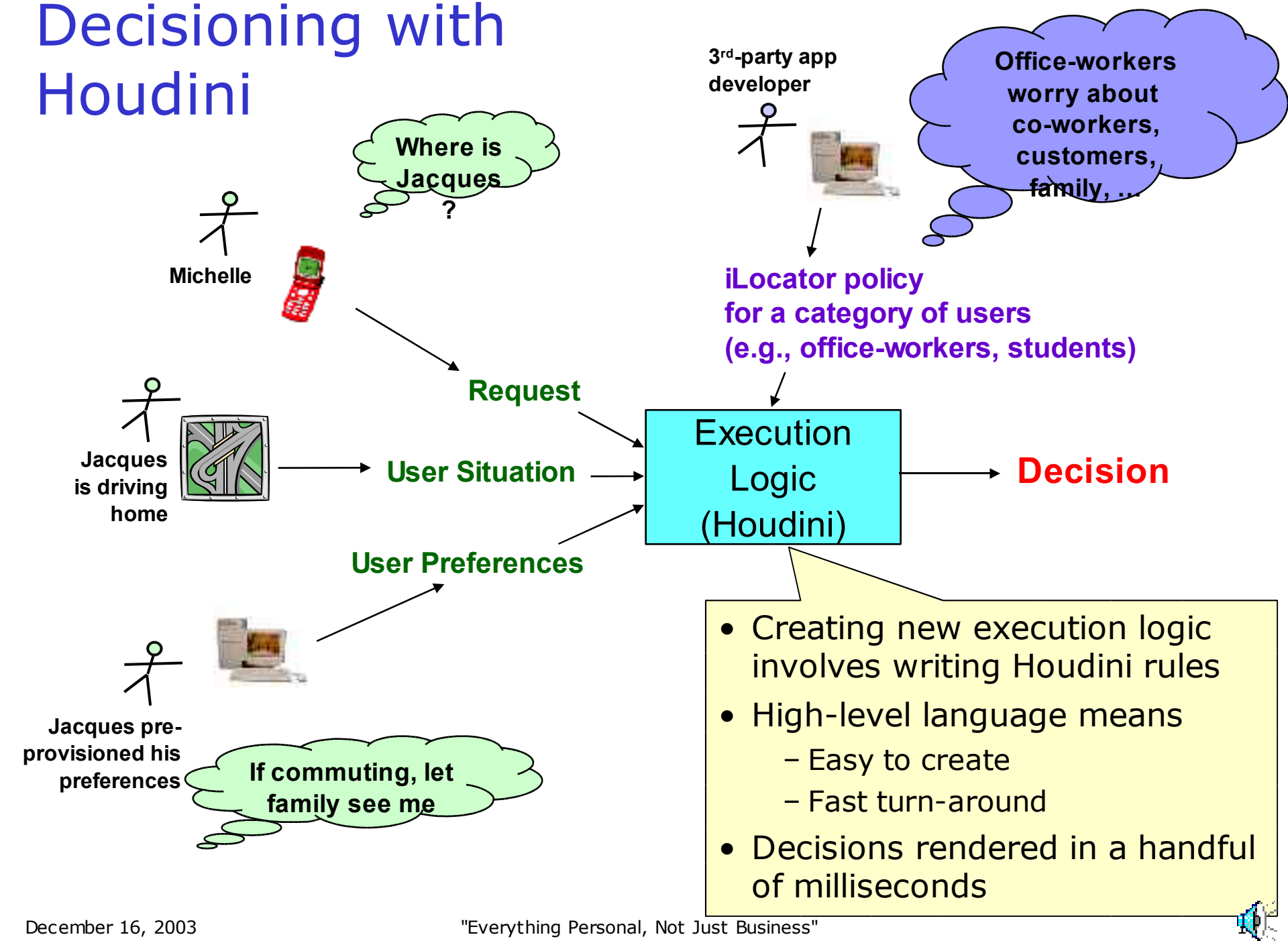
[see our paper in Mobile Data Mgmt.'04]

- Preliminary performance numbers for Houdini
  - Focus on engine (not count loading of rules, etc.)
  - Sun UltraSPARC-IIe with 1 Gb main memory
- Two kinds of rulesets considered
  - Synthetic: conditions, actions exercise different data types
  - "Selective Reach Me" prototype example (about 30 rules, some with lists)
- Results
  - ***Up to 41K rule tests/fires per second***
    - 41K: one condition per rule, scalars only
    - 19K: one condition per rule, list of record in condition, list action
    - 17K: three conditions per rule (all tested), scalars only
    - 11K: three conditions per rule, list of record in condition, list action
  - Selective reach me: ***Average response time per decision 2 to 3 millisecs***
    - 2.5 msec: Subscriber has 2 devices (89 rule instantiations)
    - 2.8 msec: Subscriber has 3 devices (105 rule instantiations)
    - 3.7 msec: Subscriber has 5 devices (150 rule instantiations)
- Presumably, substantially faster than Rete-based systems (e.g., ILOG)
  - Rete requires large main-memory data structures…

Houdini: Based on toolkit approach

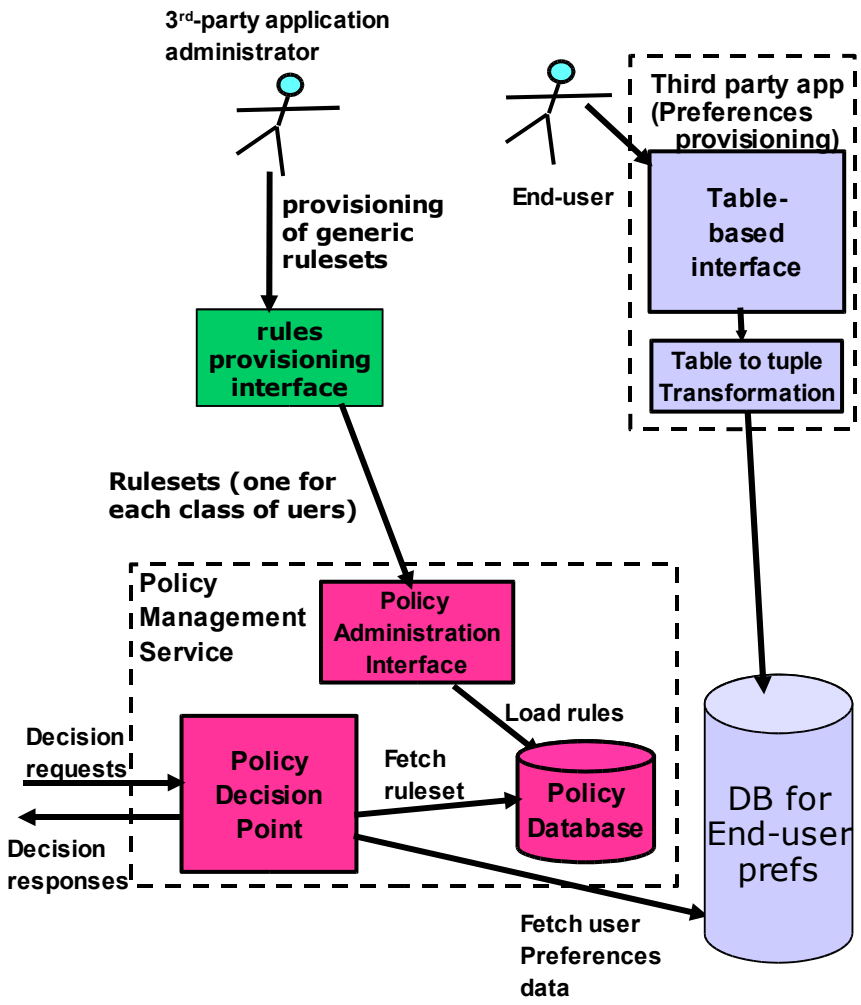


Decisioning with Houdini



- Creating new execution logic involves writing Houdini rules
- High-level language means
  - Easy to create
  - Fast turn-around
- Decisions rendered in a handful of milliseconds

Supporting Self-Provisioning of Preferences



- Forms are presented to end-user

What times are you "working"?
Monday from ____ **to** ____
Tuesday from ____ **to** ____
. . .

- Preferences data stored in database
- Preferences data retrieved by rule with query in action

```
if true
then scheduled_context :=
        SQL("select start_time, end_time, context_value
             from pref_db_scheduled_context
             where subscriber = $1,
            "Rick");
```

- "Generic rule" that can use pref data

```
If time_between(current_time.time,
                ?scheduled_context.start_time,
                 ?scheduled_context.end_time)
  & . . .
then Context = "?scheduled_context.context_value"
```

Important next step: a framework to auto-generate forms, db schema, and generic rules from single spec

Selected Related Work

- Current products
  - Personalization: typically value-driven, or auto-learning
  - Privacy: typically "all-or-nothing", perhaps by buddy list
    - Emerging standards: XACML (specify access control), P3P (to describe privacy policy)

- Policy-enablement
  - Decision trees – become too intricate
  - Rules ala IETF standards: no chaining
    - XACML also follows no chaining approach
  - Rules engines such as OPS 5, ILOG, CLIPS, IBM CommonRules
    - More expressive than Houdini, but slower
    - Typically use Rete algorithm
- Personalization research
  - Context toolkits [Dey et al '99, '00]
    - Separation of context info collection and processing logic
  - Semantic e-wallets [Gandon and Sadeh '03]:
    - Distributed, agent-based architecture
    - Use OWL ontology language and rules to capture user preferences

Conclusions

- Houdini framework: an approach to personalizing web and converged services appropriate for mass deployments
  - High-speed rules engine to capture/execute on policies
  - Forms-based self-provisioning of preferences
  - Easy to support different applications and different categories of users
- Next steps
  - Develop learning algorithms on top of Houdini language
    - Houdini ruleset structure with "intermediate variables" can provide useful outer structure for learning
  - Preferences and policies will be spread through the networks
    - Develop theory of "federated" policy management
    - First step in Policy'03 workshop

Further questions?
Please contact Rick Hull
hull@lucent.com
http://db.bell-labs.com