# *eSERL*

# Feature Interaction Management in Parlay/OSA Using Composition Constraints and Configuration Rules

**Alessandro (Alex) De Marco, Ferhat Khendek**

Dept. of Electrical & Computer Engineering
Concordia University
Montreal, Canada

# Outline

1. Introduction
2. eSERL Language Enhancements
3. Validation Algorithms
4. Implementation + Case Study
5. Conclusion

# Introduction

## Trends

- Personalization
- Added-value through service composition

## Next-generation Networks

- “Everything over IP”; IP Everywhere
- Enhanced Multimedia & Signaling Capabilities

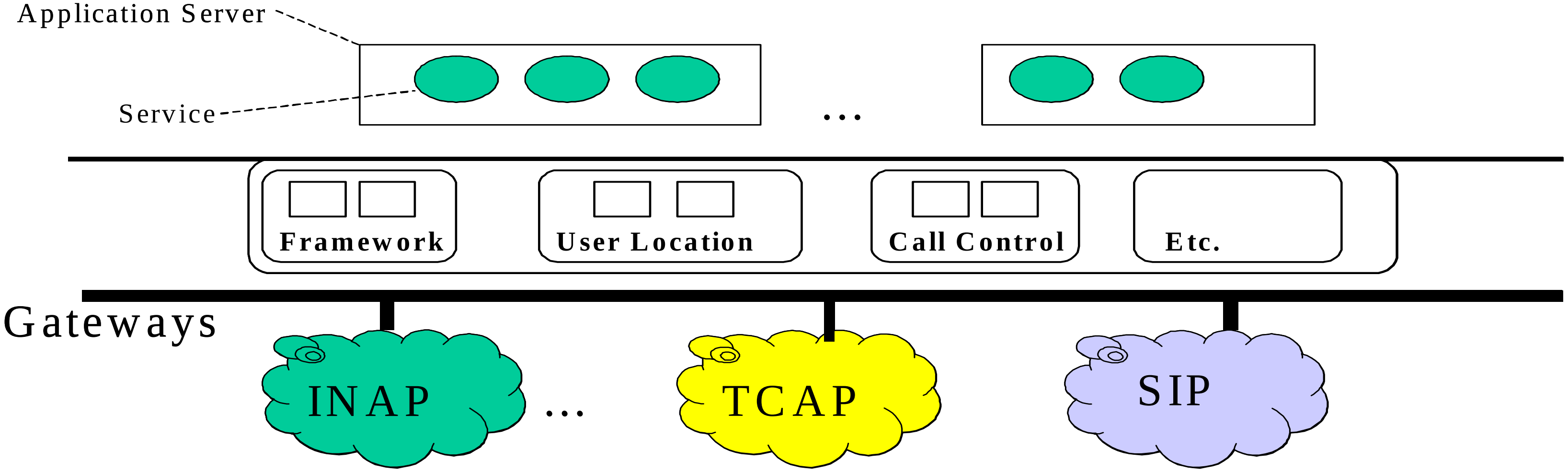
## Parlay/OSA

- 3GPP API for secure, open access to NG Networks
- Technology-agnostic

## SERL

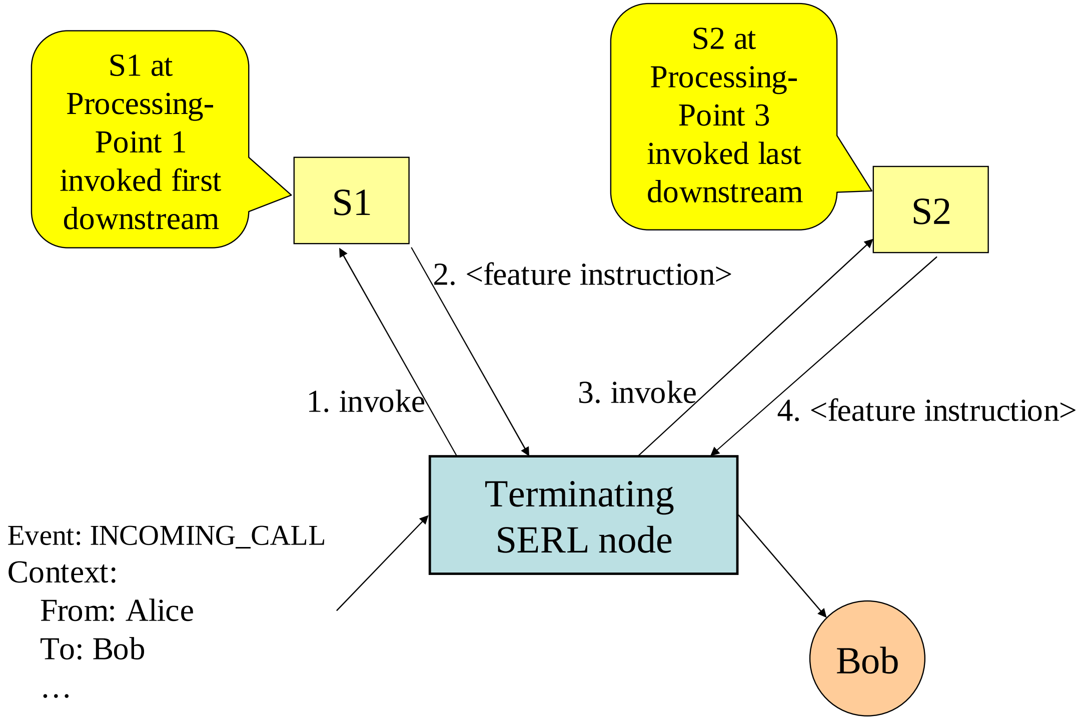
- Service Execution Rule Lang. & Framework
- No FI detection; Only application of resolutions

# Parlay/OSA



- Open Service Access standard adopted by 3GPP
- Access to core networks through secure framework
- Not just Call Control, but Mobility, IM, more
- Technology-agnostic

# SERL



Service Execution Rule Language

3 Internet Drafts in 2001 (Ericsson)

FIM intercepts events, matches & applies rules to trigger services

No FI detection or avoidance capabilities

No known implementations

# eSERL: Enhanced SERL

- Language Extensions
  - Service Objects (named with I/O params)
  - Composition Constraints
  - Configuration Rules
- Feature Grouping Criteria
  - Distinguish between routing & screening

# Composition Constraints

- SUSC context: 1 user, 1 app server
  - Service interactions are known/detected *a priori*
  - Use any detection techniques
- Experts define service composition and inter-working constraints
  - Explicit vs. implicit constraints
  - Mutex, Order, Data Inter-working

# Configuration Rules

- End-user requirements for their service behavior
  - Expressed as condition-action rules
  - Conditions relate to events
  - Actions affect services, or events
- Backwards-compatible with SERL

# Operational Context

- Experts
  - Define constraints for all services in a system
- End-users
  - Write configurations to compose and personalize services
  - Deploy configurations
- System
  - Validates* configuration (offline tool)
  - Intercepts events, matches & applies rules (runtime Feature Interaction Manager*)

# Abstract Example

Participants: Julie (the driver) and her car

```
If (INCOMING_CALL or OUTGOING_CALL) {
        Invoke CS(screening party: car)
        If (response from car: Julie is AVAILABLE) {
            Invoke ID(“warn that call may be di
        }
}
If (Session.CallExists(Julie)) {
        If (INCOMING_CALL from car and car says Julie is BUSY) {
            Invoke ACB // which terminates call, re-establishes later
        }
}
```

High-speed,
heavy traffic

# Abstract Example

Is this user-defined configuration “valid”?

# Validation

- Check configurations against constraints
- Guaranteed behavior
  - To the degree with which the expert is confident with the completeness and consistency of constraints

# Acceptable Compositions

- ‘Acceptable’ = All compositions except those in violation of constraints
- Completeness Assumption
  - Approaches a “complete-set”
- Consistency
  - Worst-case: no compositions allowed
- Approach depends on expert experience, tools, maintenance of rule-base

# Detect Constraint Violations

- Simple: 1 rule, several actions
  - Order or mutex violation (composition)
  - I/O params set (data inter-working)
- Complex: n rules, >0 actions for each
  - Rules satisfied simultaneously by event? i.e. Do conditions overlap?
  - If overlap, then

    compose the actions, and

    check for violations as for simple case

# Pair-wise Rule Comparison

```
For rule1, where rule1 is a Configuration Rule
    For rule2, where rule2 is a Configuration Rule and not rule1
        If rule1.condition and rule2.condition overlap then
            If rule1.action composed with rule2.action is
               not in set of acceptable compositions then
                Configuration Rule Module is invalid
```

# Rule Overlap

## Calculating overlap

- Polynomial time solution, $O(n^k)$, if values for variables are discrete, finite, and ordered (D. Wang et al., IP firewall study)
- Parlay/OSA API methods, events meet criteria

## Example 1: Overlap: Yes

C1 := {“my location is home”}

C2 := {“caller is bob@school.com”}

## Example 2: Overlap: Maybe … syntax vs. semantics

C1 := {“my location is school” AND “caller is alice@home.com”}

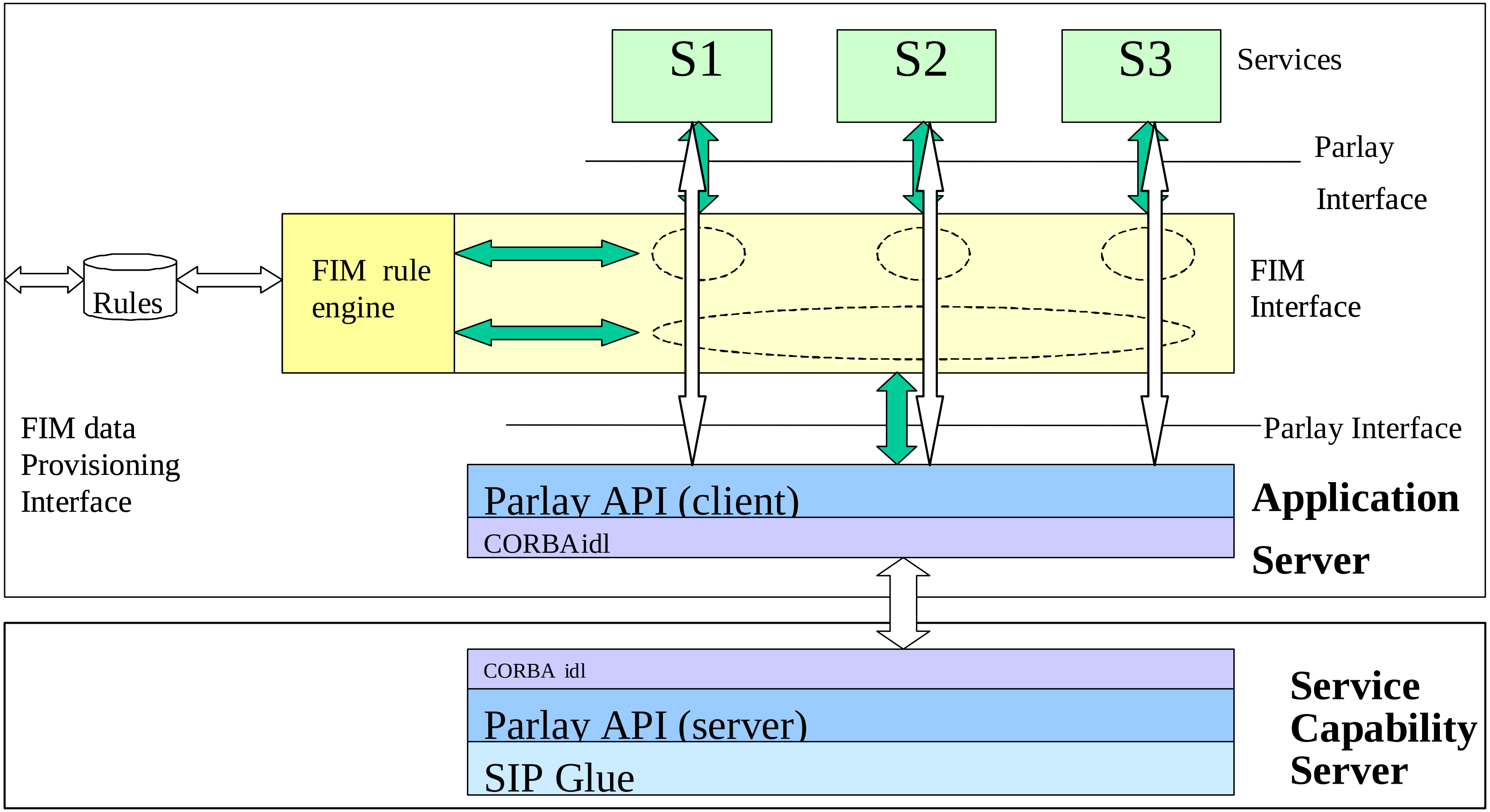
C2 := {“my location is office” AND “caller is sales@company.com”}

# Rule-Action Composition

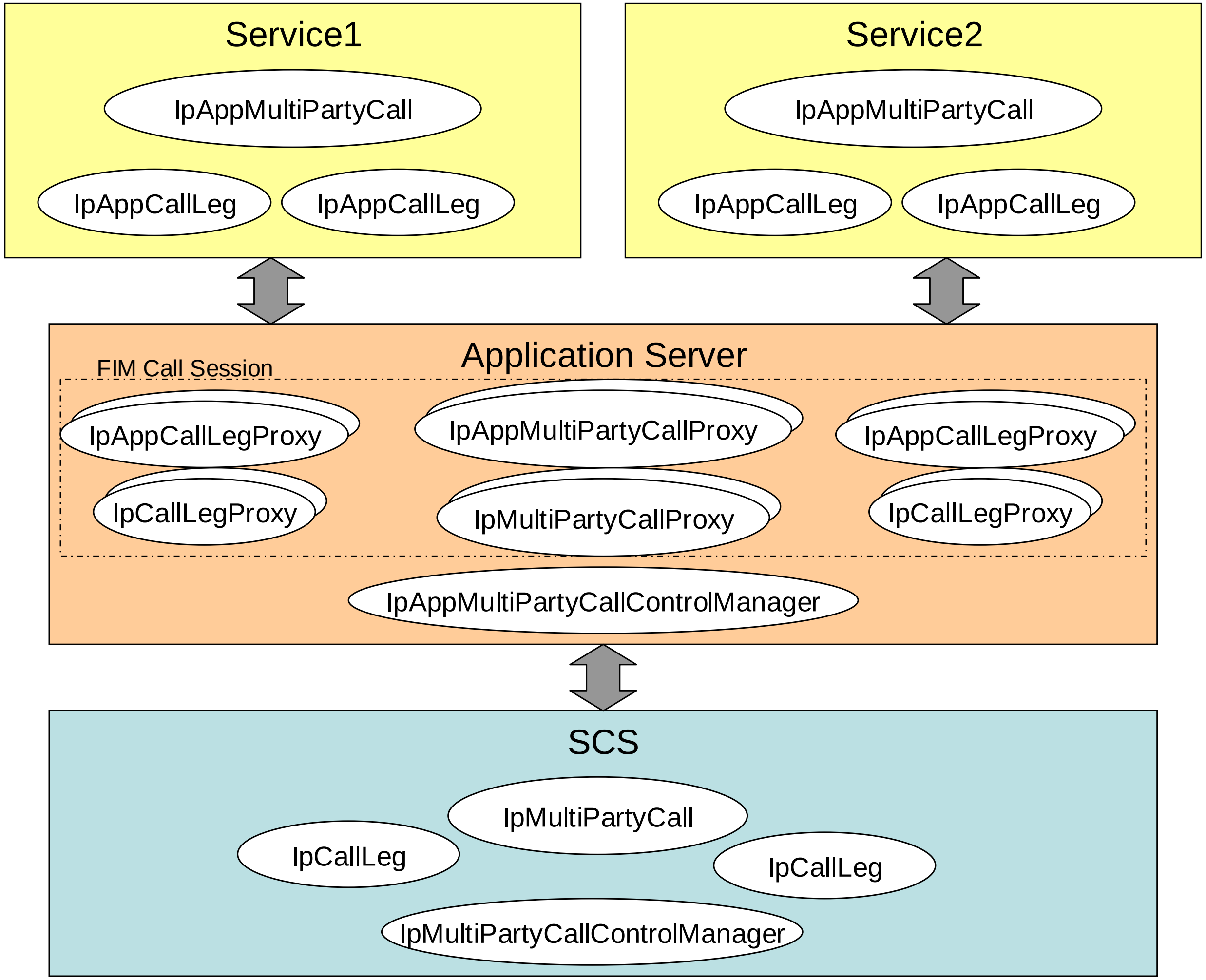
- Composing Actions, order is important
  - Compare Processing Points
  - Compare priorities of rule actions
- A single configuration may specify many compositions.
  - If one is invalid, the whole configuration is rejected.

# Implementation

## Positioning of a FIM in the architecture



# Session & Proxy Objects (+ Event Translation)



# Julie Jones and the Family Car

Incoming/Outgoing calls to/from driver - Julie Jones

- Screening by car (CS)
- If screening passed, warning (ID)

Call in-session

- Julie becomes BUSY, save & disconnect (ACB)

ACB waiting

- Julie becomes AVAILABLE, retry (ACB, [CS, ID])

Location too far from home

- Instant message to Mom (ID)

# Results

- Hand-written rules in terms of Parlay/OSA events.
- Implemented tools to validate rules against the system constraints.
- Implemented test architecture, including FIM.

# Contributions

- Generic framework for service personalization and composition while managing FI
- Guarantee, to a certain degree, on composed service behavior provided there are no constraint violations
- Design & implementation in Parlay/OSA context

# Future Work

- Multiple users, Multiple Servers
- Activation Rules
- Non-monotonic extensions due to system constraint changes
- Framework for writing rules with 3rd party “theme-based” rule templates and wizards
- Composition Constraints = 3rd party services

# Thank you.

- Questions ?