

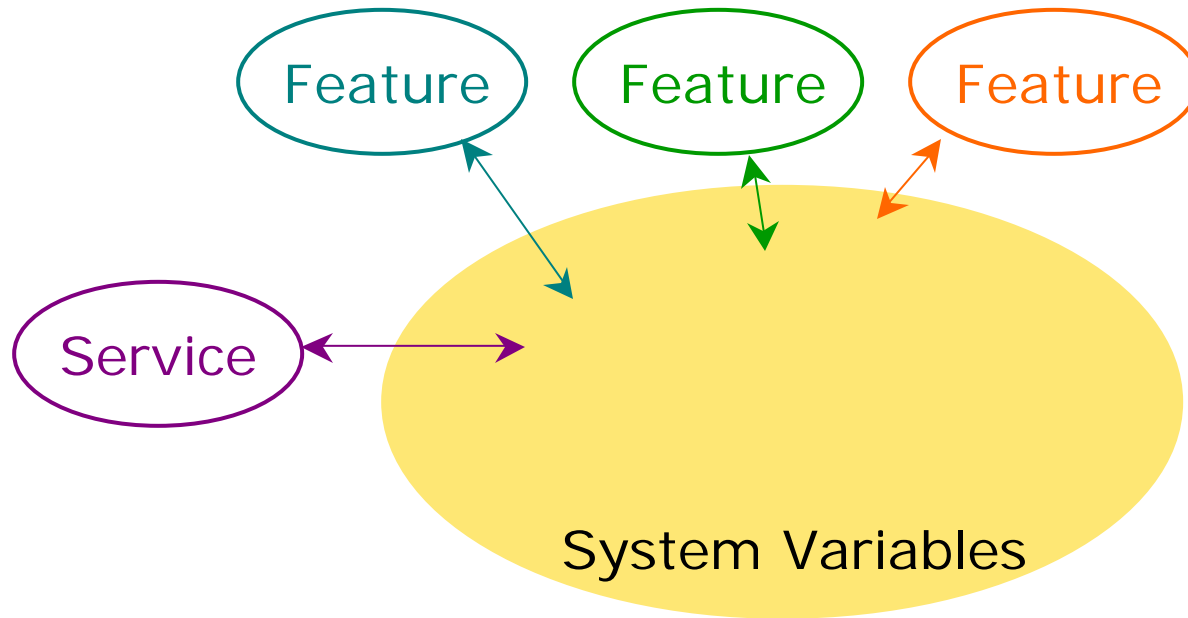
Features, Policies and Their Interactions

Joanne M. Atlee
Department of Computer Science
University of Waterloo

Outline

- **Features and Feature Interactions**
- **Policies and Policy Interactions**
- **Strategies for addressing the Interaction Problem**
- **COURRT - COordinating User pReferences at Run-Time**
- **Summary**

Features

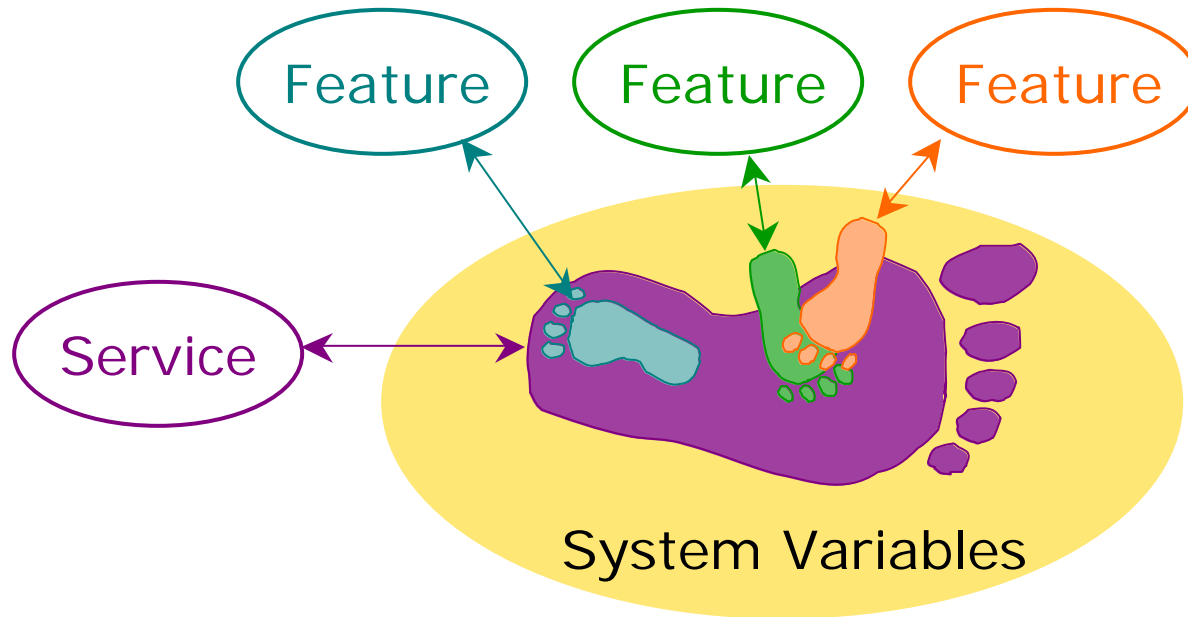


Service - an application's **core functionality**

Feature - **incremental functionality** to an existing system

Ideally, we would like to view features as **separate concerns** to be implemented **modularly** as independent increments to the system

Feature Interactions



The problem is that features are not completely independent of each other in that they manipulate **shared system variables**

Feature Interaction - a **change** in one feature's behaviour due to the presence of another feature

Example Feature Interactions

Call Forward No Answer vs. Voice Mail

- both are enabled by “no answer”, but it makes no sense to activate both
- should activate call forward if goal is to reach a human
- should activate voice mail if goal is to reach dialed party

Originating Call Screening vs. Call Transfer

- which call is screened?

911 vs. Hold

- cannot put 911 operator on hold
- does this constraint disable all features that use a form of Hold? (e.g. Call Waiting or Three-Way Calling)

Classes of Feature Interactions

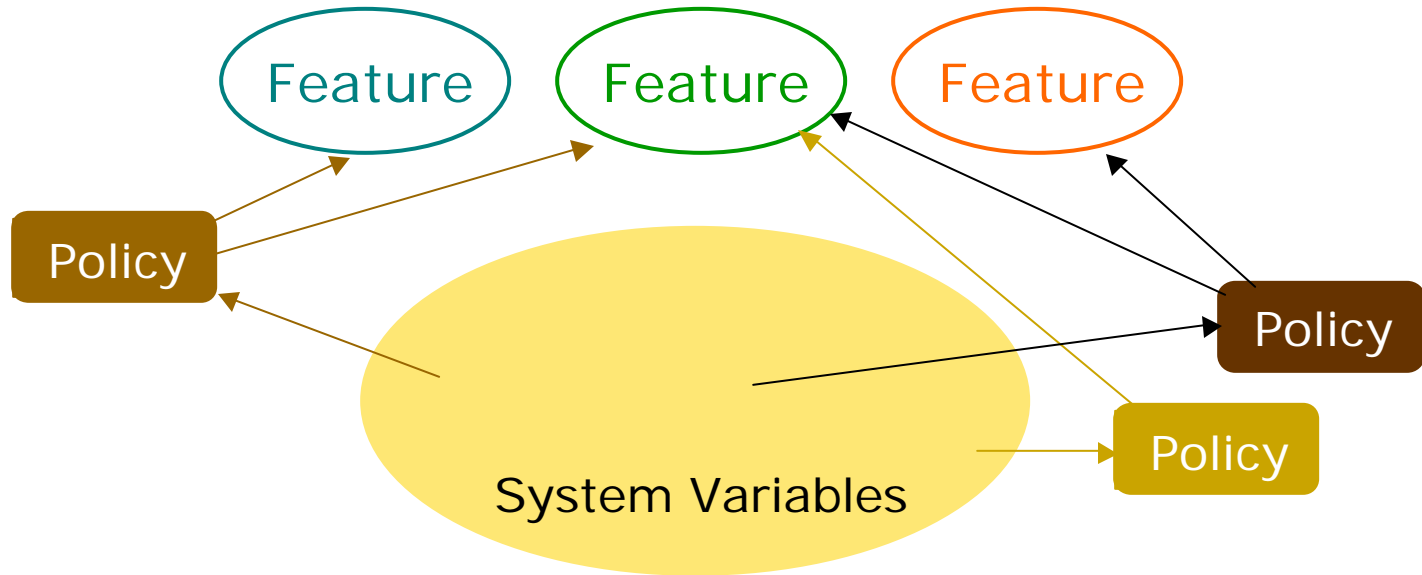
Individual features affect system behaviour by

- assigning new values to variables (e.g, Call Forward)
- constraining variables values (e.g., 911)

Feature combinations interact when the individual features' actions conflict

- variable assignments are inconsistent
- new variable assignment violates constraints
- new constraint is violated by current variable values
- constraints are unsatisfiable

Policies

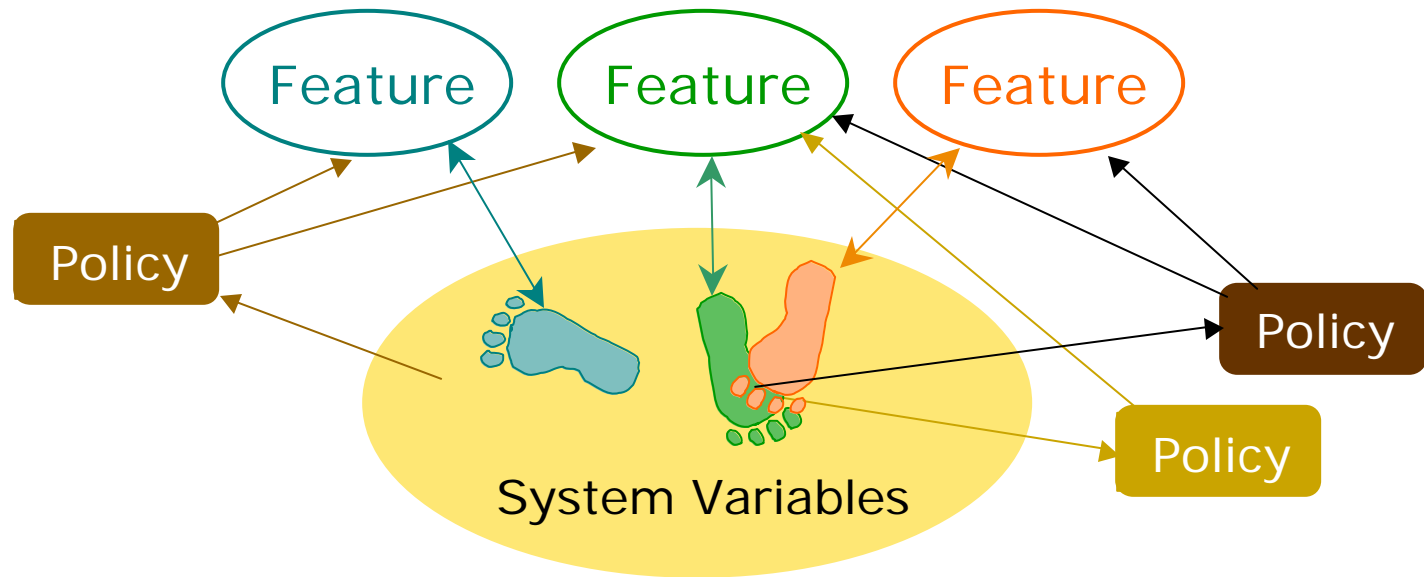


Policies - information that modifies system behaviour

- **data** (cf. program)
- **manipulate features** (cf. manipulate variables)
- specify **long-lived goals**/constraints

Features are no longer invoked and stimulated only by users or call states

Policy Interactions



The problem is policies are not completely independent of one another because they may

- read and react to the **same variable values**
- manipulate the same **shared features**

Policy Interaction - a **change** in one policy's behaviour due to the presence of another policy

Example Policy Interaction

Features affected

- Call Forward on No Answer
- Voice Mail

Policies

- redirect long distance calls to forwarded number
- redirect calls from management to voice mail

Interaction

- what if a manager calls from long distance?

Classes of Policy Interactions

Individual policies affect system behaviour by

- invoking features
- constraining feature invocation

Policy combinations interact when the individual policies' actions conflict

- feature invocations are inconsistent
- new feature invocation violates constraints
- new constraint is violated by currently active features
- constraints are unsatisfiable

Approaches to the Interaction Problem

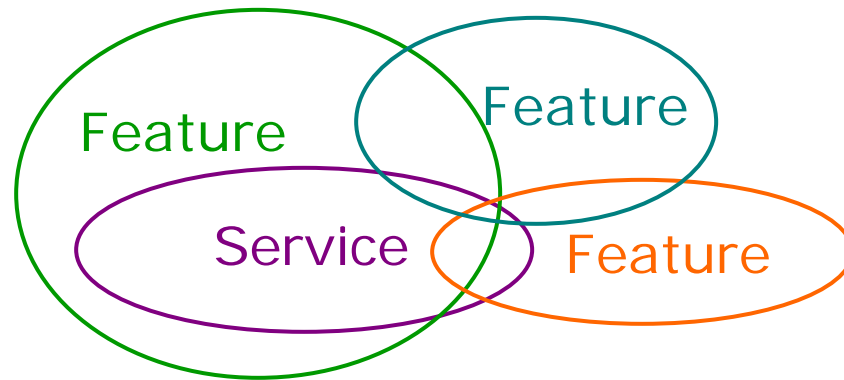
Eliminate interactions during design either by re-designing individual features/policies or by specifying how feature combinations should behave.

Prevent interactions via architectural constraints, thereby coordinating the features'/policies' access to shared resources.

Resolve interactions at run-time, thereby applying corrective action only when an interaction actually occurs.

Design-Time Approaches

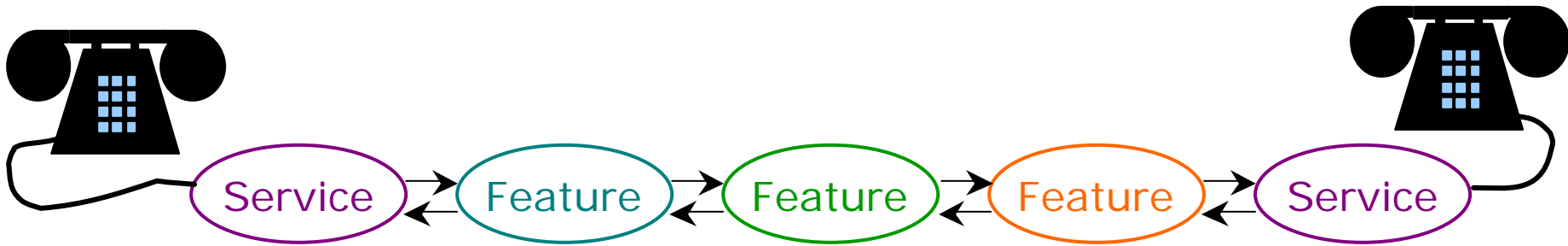
Eliminate interactions during design either by re-designing individual features/policies or by specifying how feature combinations should behave.



- + can realize ideal **feature-specific resolutions** to interactions
- system eventually becomes **difficult to maintain and extend**
- provides **one-solution-for-all-users resolutions**

Architectural Approaches

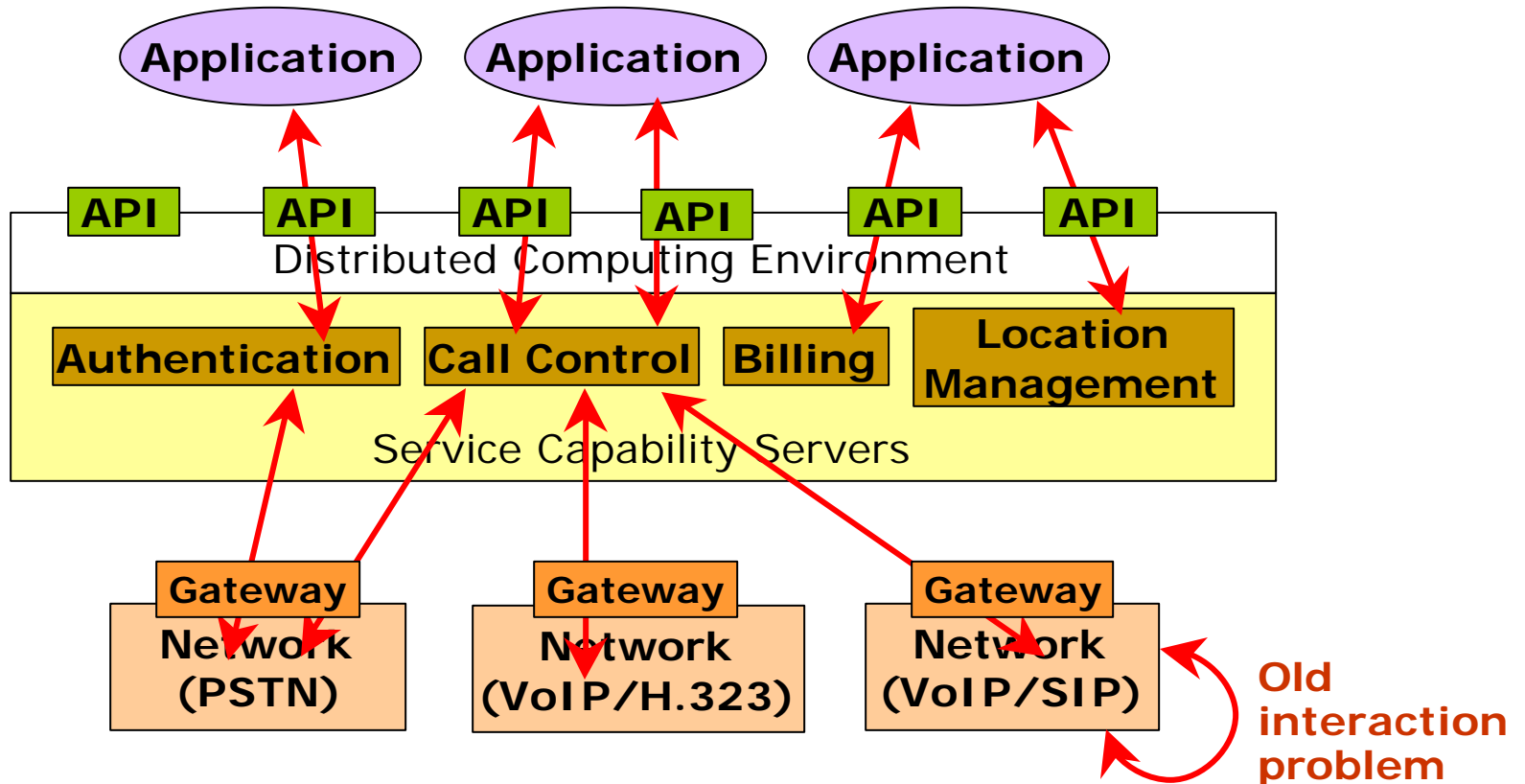
Architectures can coordinate and restrict features'/policies' access to shared resources.



- + messages act as **tokens** that **serialize** features' actions
- + **placement** of features in sequence realizes **priority scheme**
- **may not resolve constraint interactions**
- implements **one resolution** strategy - precedence

Architectural Approaches

Open System Architectures may provide **network-independent environments** on which to develop applications, but they do nothing to ease the interaction problem.



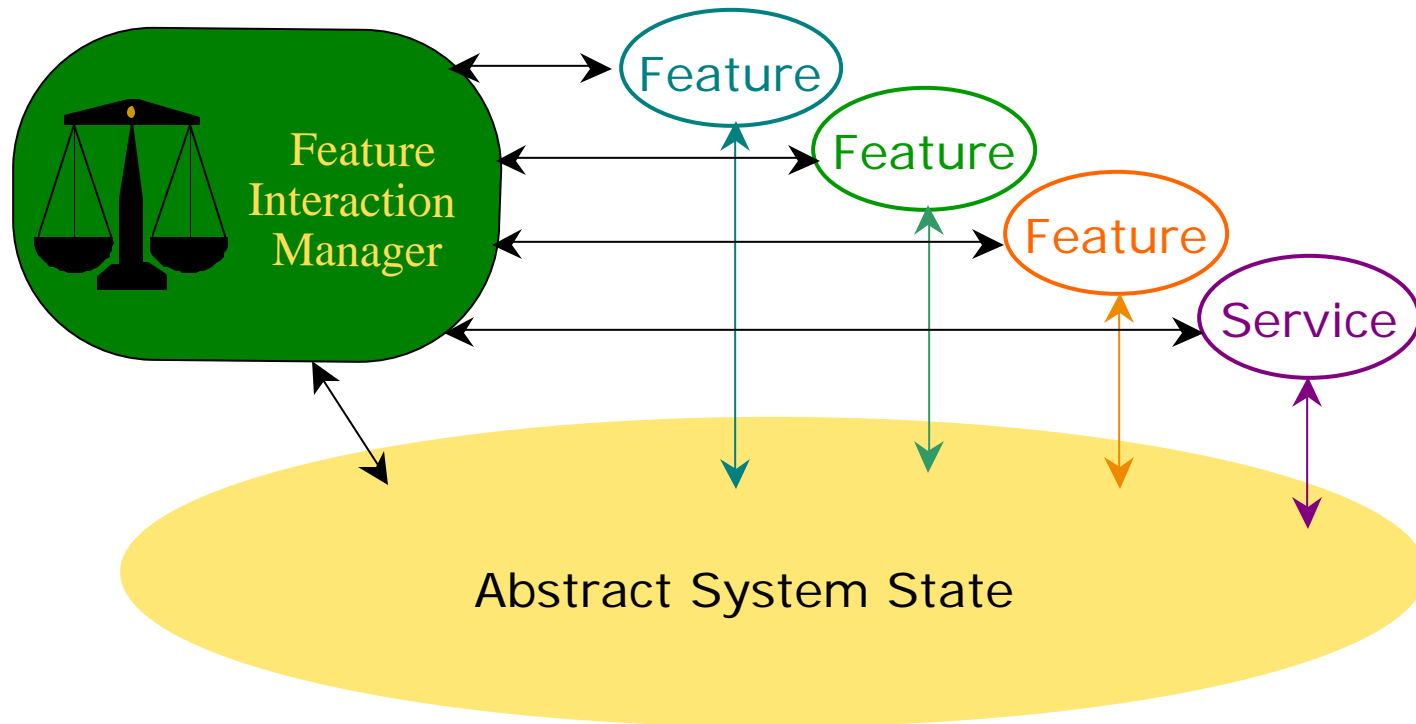
Run-Time Approaches

Detect and resolve interactions at run-time

- + Only applies corrective action in the event of an **actual interaction**
- + **Need not over-constrain** feature to avoid **rare interactions**
- Must detect and resolve interactions **on-the-fly**
- Must not require **proprietary knowledge** of features

COURRT

COordinating User pReferences at Run-Time



- Features are implemented as **independent agents**
- **Application-independent** Feature Interaction Manager agents are responsible for **detecting** and **resolving interactions**
- Resolution strategies are **encapsulated** in the FIMs

COURRT Abstract System State

The way it works - We model an **abstraction** of the system state as a set of **relational variables**.

- Each **abstract variable** is a **relation**.

Connection

| Conn ID | User |
|---------|------|
| C1 | U1 |
| C1 | U2 |

Screening List

| Subscriber | Screen |
|------------|--------|
| U1 | U2 |
| U1 | U3 |

- Each **feature constraint** is a **constraint on table entries**

ScreeningList(u1, u2) \mathbb{P}
 $\exists c . \text{Connection}(c, u1) \text{ and } \text{Connection}(c, u2)$

COURRT Features and Interactions

As a feature executes, it

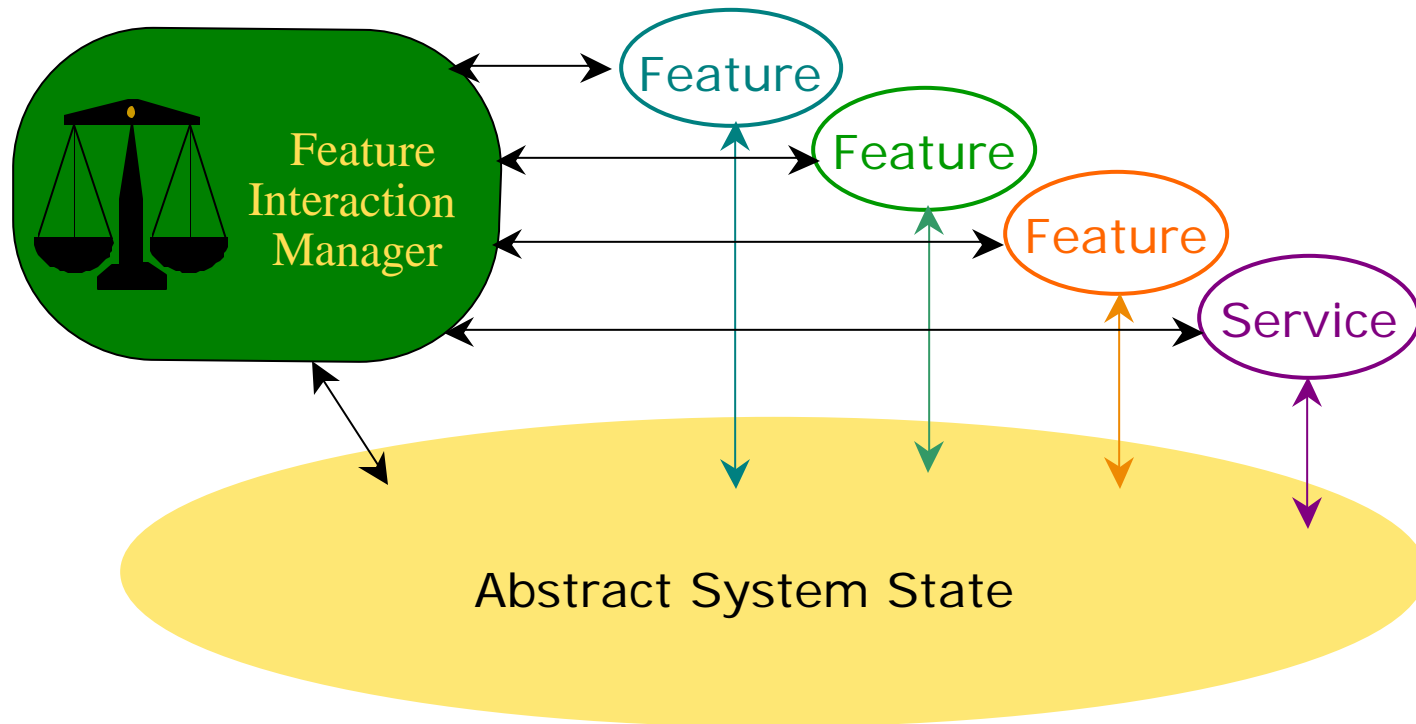
- **adds and removes elements** (i.e., rows) from relations
- **asserts and retracts constraints** on relation values

Adding a new feature to the system may **introduce new relations** (i.e., tables) to the abstract system state.

A **feature interaction** manifests itself as

- an **inconsistent assignment** to the relational variables
- a **violated constraint**

COURRT Arbitration



1. Features **declare their intentions** in terms of actions and constraints on **abstract variables**
2. A local FIM **detects interactions** and **arbitrates resolutions** according to some **resolution strategy**
3. Features execute resolution

COURRT Reasoning

Twist - An interaction **resolution** may **introduce an inconsistency** into the database.

- The actions of a higher priority feature may violate a constraints of a lower priority feature (e.g., one cannot screen calls from emergency services)
- A constraint may be unsatisfied when it is introduced (e.g., one can add currently connected parties to a screening list)

This means that the system and features must be able to continue to **function when the database is inconsistent**.

Thus, a **feature interaction** is actually any

- **new inconsistency**
- **violation** or **re-violation** of a constraint

Resolution Strategies

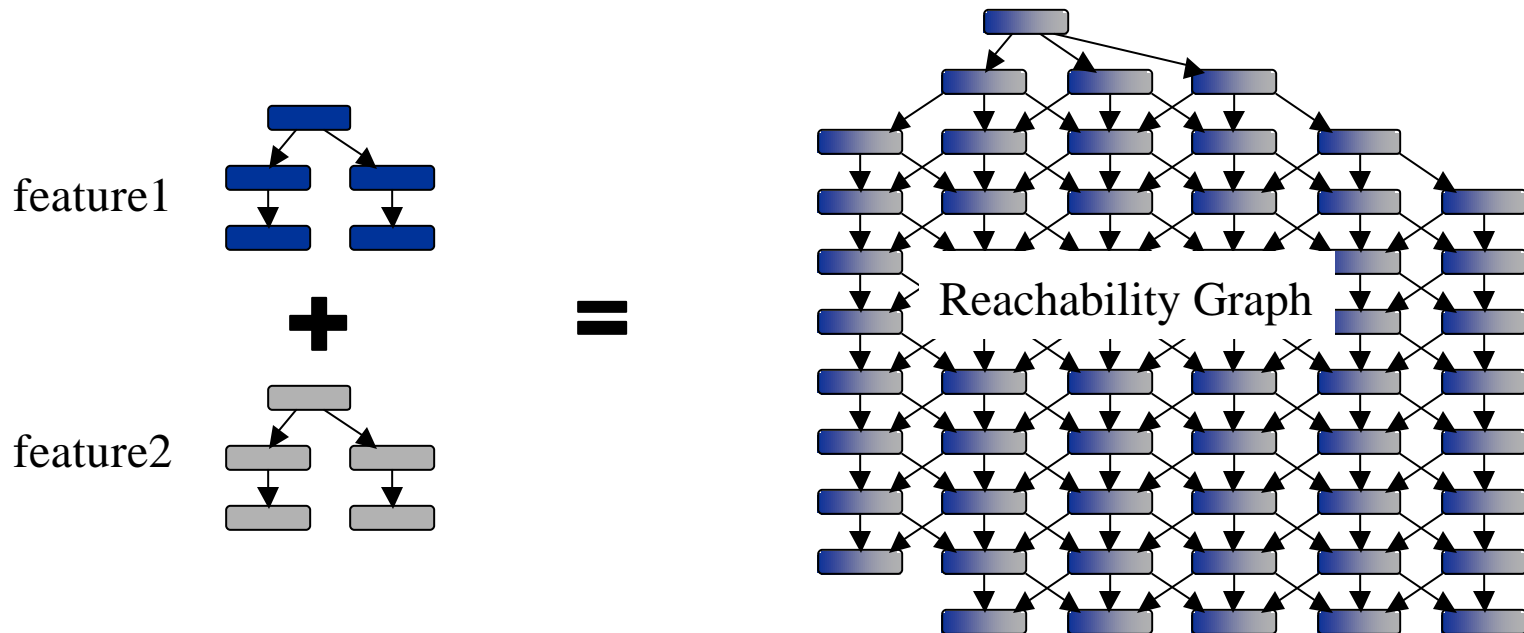
Big Question - which resolution strategies are generally effective and produce predictable behaviours?

- abort a feature (e.g., based on priorities)
- abort a rule
- abort a feature rules' individual actions, constraints
- specify exceptional rules, behaviour
- serialize conflicting accesses to shared resources
- specify resolution policies
- negotiate a compromise
- do nothing
- combine the above techniques

COURRT Analyzer

We are implementing an analyzer to test the **usefulness** and **predictability** of these various resolution strategies.

- analyzes all **executions** of a combination of features
- applies an encapsulated **resolution strategy** to interactions
- reports **interactions** and their **resolutions**



Summary

In order to succeed in providing **feature-rich services** in a **distributed** environment, we need

- **modular** feature development
- **feature-independent** architectures that **coordinate** features' actions and constraints
- **distributed** run-time detection of interactions
- locally **customizable** resolution strategies...
- ...that result in **predictable** behaviour

COURRT Features

Features declare their effects on the system state in **feature rules**

```
on event  
if condition  
then actions  
assert constraints  
retract constraints
```

where

event is a message or a change in variable value

condition is a predicate on variable values

actions add or remove elements (I.e. rows) from a relation

constraints on relation values can be asserted or retracted

Example (Originating Call Screening):

```
on callstate(c,analyze,u1,u2)  
if ScreenList(u1,u2)  
then add callstate(c,dead,u1,u2)
```