



SOFTWARE ENGINEERING



Run-Time Conflict Resolution for Personal Features

Joanne Atlee
Department of Computer Science
University of Waterloo

Motivation: Personal Features

- Trend towards giving users more control over their applications
 - subscriptions to telephone features
 - options for customizing game characters
 - pseudo-database interfaces to information systems
- Allow users to customize or create their own feature variations
 - specified using scripts and policies

Problem: Interactions

Users will create conflicting features

- Example: Policies to redirect calls
 - Redirect long distance calls to another number
 - Redirect calls from management to secretary
 - What if a manager calls from long distance?
- Example: Call Screening vs. Call Transfer
 - Which calls are screened?

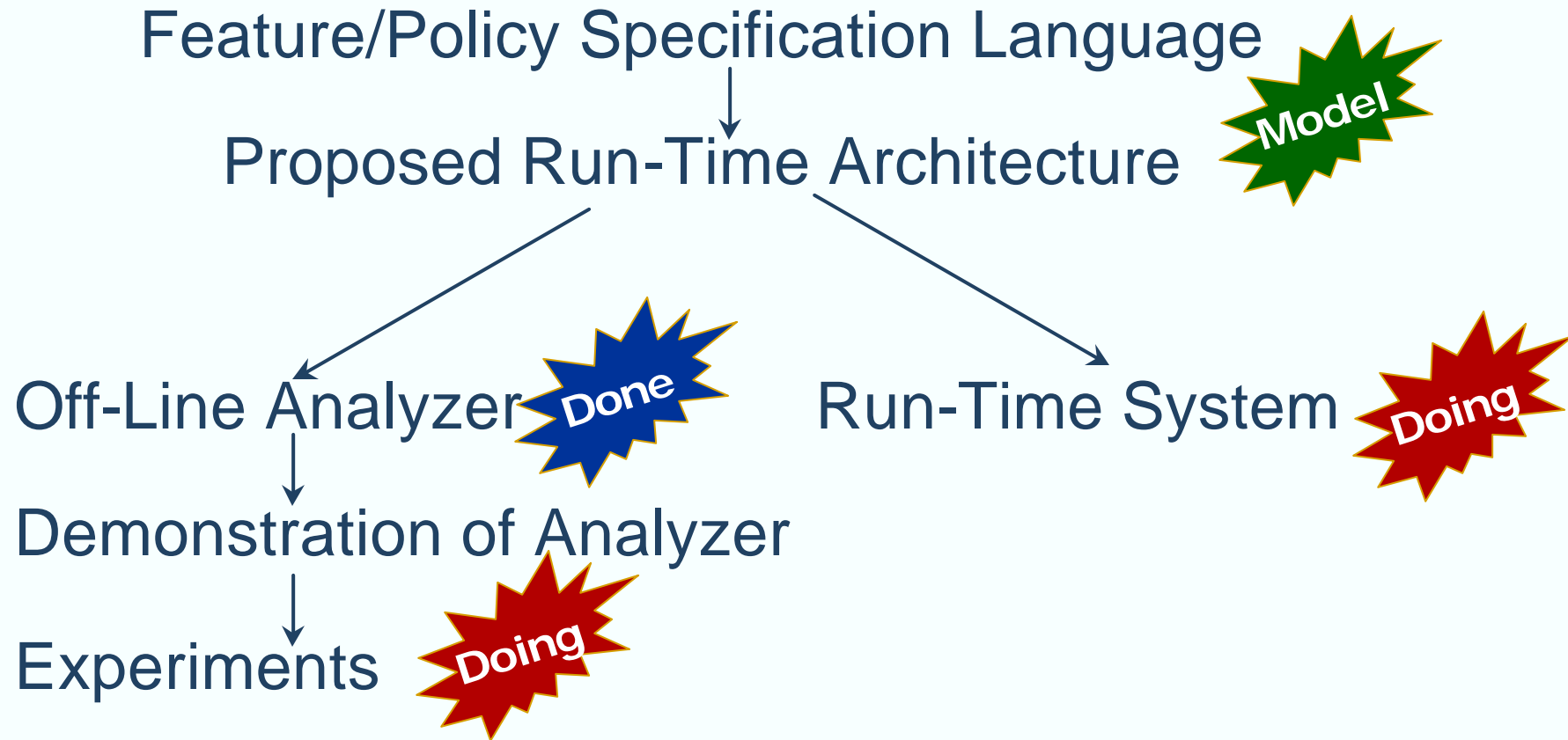
Problem: Interactions

- Users are not application experts
 - Currently, systems analysts are responsible for studying how features behaved in combination and for resolving undesired interactions.
- Users will create conflicting features
 - Need to help users detect and resolve conflicts
- Usability is an issue
 - Features and feedback must be expressed in terms that the user can understand

Conflict Resolution

- Detect and resolve conflicts at run-time
 - corrective action only taken in the event of an actual interaction
 - need not over-constrain features to avoid rare interactions
- Resolution Strategies
 - serialize features
 - impede a feature's progress
 - inhibit secondary, non-essential actions
 - query user
- Predictability is an issue
 - Resolution strategies must generate sensible resolutions

Topics

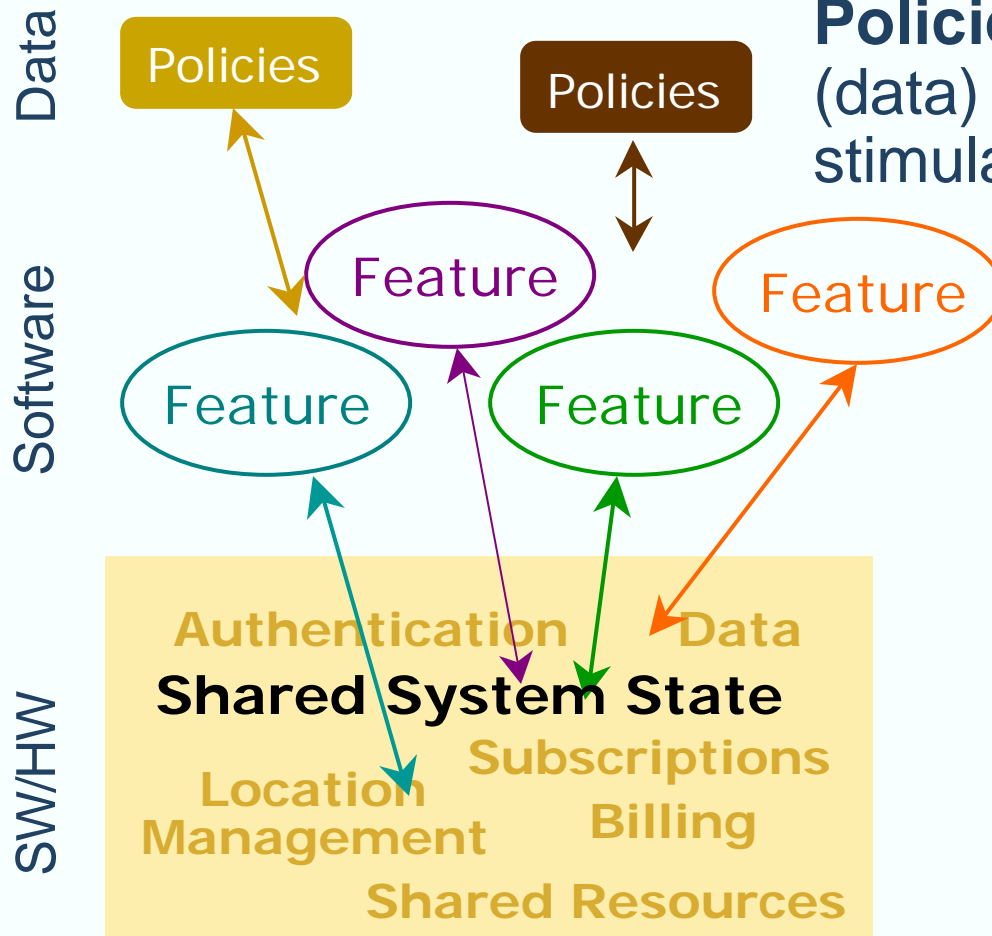


Model

Users manipulate policies

Policies - independent rules (data) that manipulate (activate, stimulate, and constrain) features

Features - independent functions (code) that manipulate shared system variables



Feature/Policy Rules

Each feature and policy is modelled as a set of conditional rules that define how the feature/policy reacts to the system state.

guard	{	on: event	<i>(message)</i>
		if: condition	<i>(predicate on facts)</i>
actions	{	then: actions	<i>(add or remove facts)</i>
		assert: constraint	<i>(formula on facts)</i>
		retract: constraint	

Example: Notify me when Annie is in her Office

if: Where(Annie,AnnieOffice)

then: +Message(Jo, "Annie in AnnieOffice")



Shared System State

The rules manipulate the shared system state, which is a database of facts, events, and constraints about user data and feature state.

- Each fact and event is a tuple (an element of a set, relation, or function):



Shared System State

The rules manipulate the shared system state, which is a database of facts, events, and constraints about user data and feature state.

- Each constraint is a named formula on facts or fact patterns

Universal quantification

Marsha cannot initiate calls, she can only receive them

NoCall(Marsha):

no c | Originator(c, Marsha)

No one on Steve's screening list may be in Steve's connection conn1

CSCon(conn1,Steve):

all u | **not** CS_List(Steve,u) **or not** Connection(conn1,u)

More Feature Rules

Call Screening

If a caller subscribes to call screening, keep screening list entries out of connection

if: Connection(?c,?u) **and** CS_Subscribe(?u)

and: not CSCon(?c,?u)

assert: +CSCon(?c,?u)

Existential quantification
(bound throughout rule)

Call Forward to Location

Forward my calls to my current location

if: Call(?c, ?t, allocate, ?o, Marsha)

and: Where(Marsha,?p_room) and Owner(?p,?p_room)

then: -Call(?c, ?t, allocate, ?o, Marsha)

then: Message(Redirect,?c, ?o, Marsha, ?p)

More Feature Rules

Boss in Room

If a higher ranking person is in room, forward to voice mail calls destined for me

if: Call(?c, ?t, allocate, ?o, Marsha)

and: PhoneInRoom(?t, ?t_room)

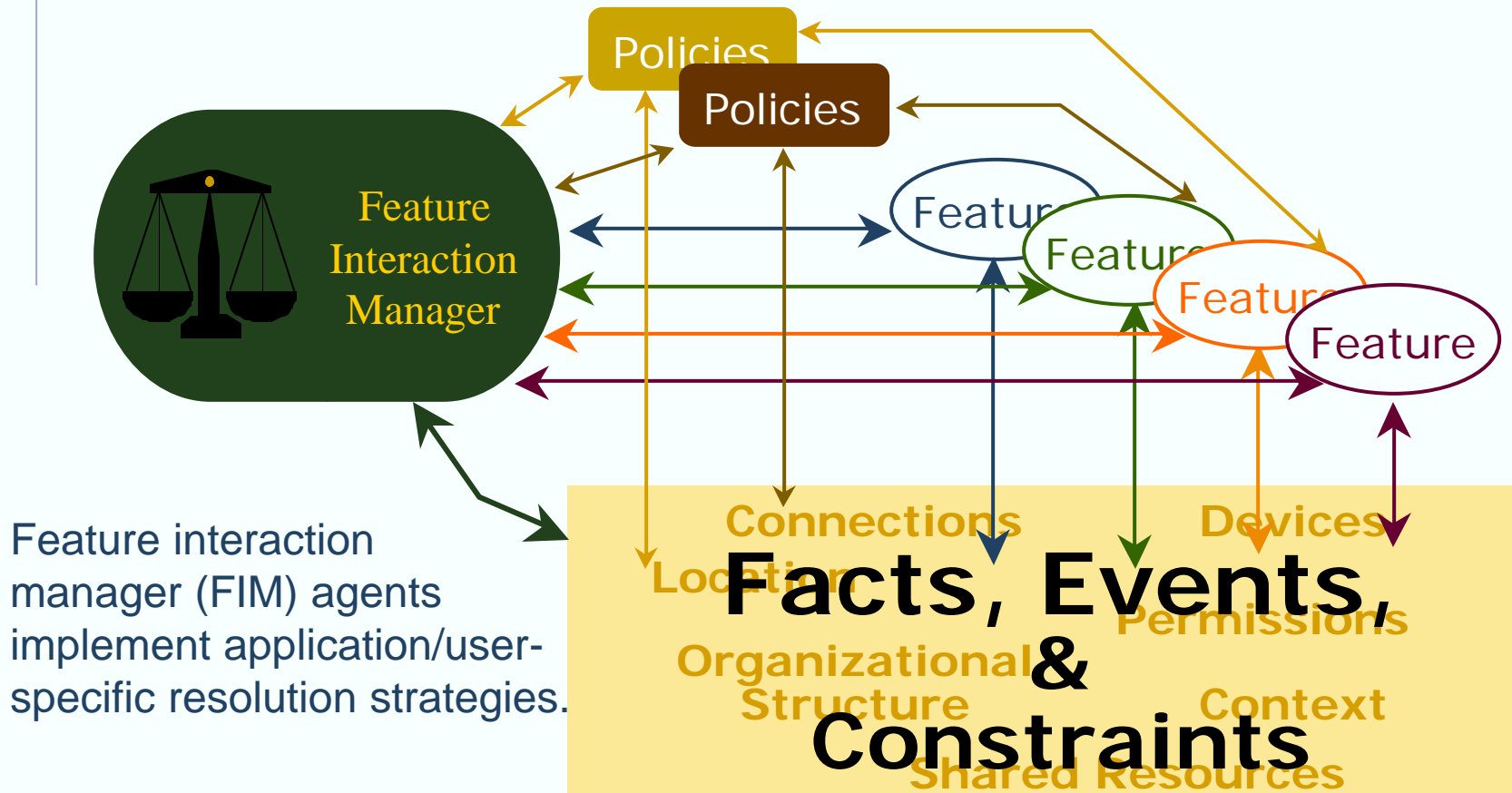
and: Where(Marsha, ?t_room) **and** Where(?p, ?t_room)

and: Rank(Marsha) < Rank(?p) **and** Rank(?o) < Rank(?p)

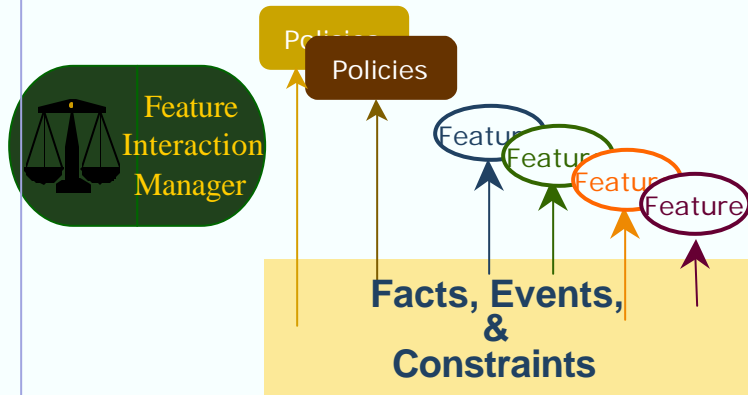
then: -Call(?c, ?t, allocate, ?o, Marsha)

then: Message(Redirect, ?c, ?o, Marsha, MarshaVM)

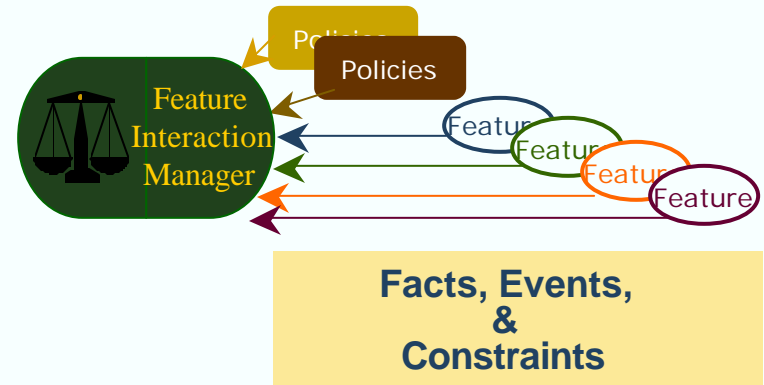
COordinating User-preferences at Run-Time (COURT)



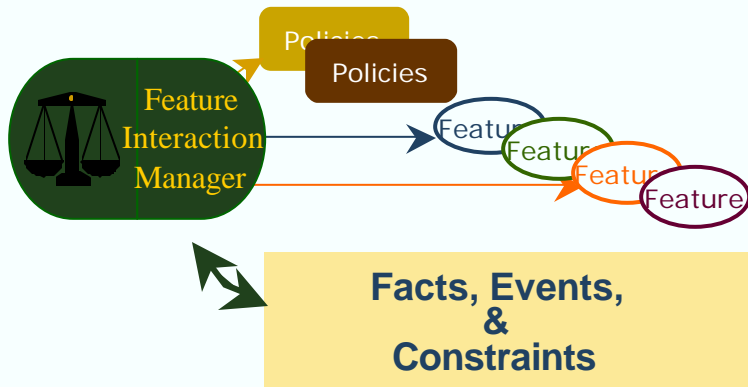
COURT Arbitration



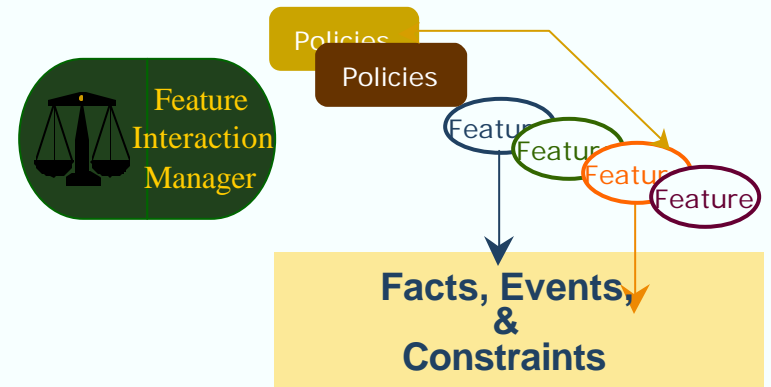
1. Features/policies monitor system



2. Features/policies declare intentions (changes to facts and constraints)



3. FIMs detect interactions and arbitrate resolutions



4. Features execute resolutions

COURT Interactions

Features/policies interact when the feature's/policy's individual actions conflict

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

COURT Interactions

Features/policies interact when the feature's/policy's individual actions conflict

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

These interactions should be detected and resolved

COURT Reasoning with Inconsistencies

Twist - Tolerating an interaction introduces an inconsistency into the system state (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.

COURT Reasoning with Inconsistencies

The only interaction classes we try to detect are

- inconsistencies among new predicates

+Rel(x,y) -Rel(x,y)

+Funct(x,y) -Funct(x,y)

+Funct(x,y) -Funct(x,z)

- violations or re-violations of a constraint

Let State be the current system state

State' be the system state after some rule

State' contain constraint " i.C(i)

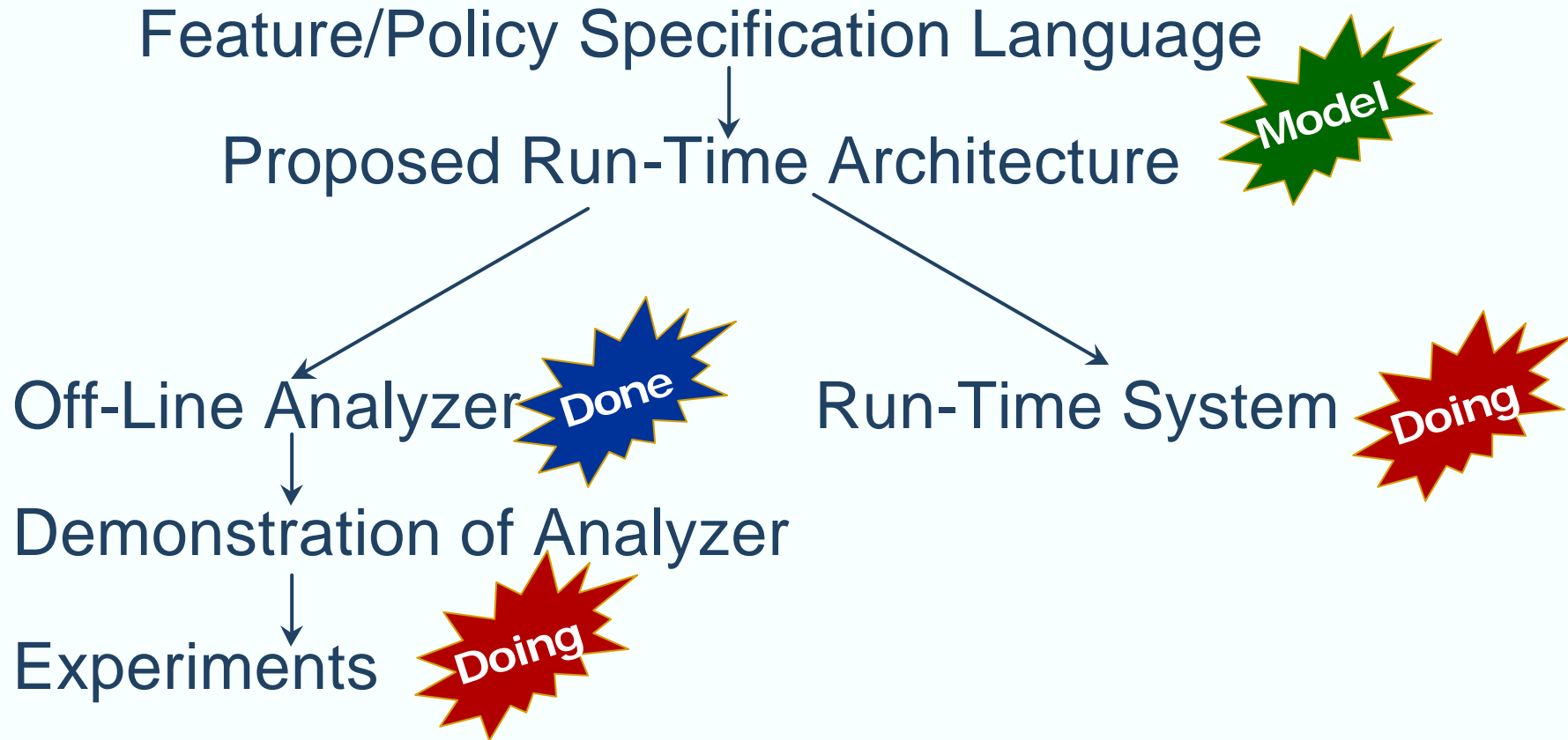
$\exists i . (\text{State} \triangleright C(i) \text{ and } \text{State}' \not\triangleright C(i))$

COURT Resolution

Big Question: Which resolution strategies generally produce predictable behaviours?

- abort a feature/policy (e.g., based on priorities)
- abort a rule
- abort a rules' individual actions, constraints
- specify exceptional rules, behaviour
- serialize conflicting accesses to shared resources
- specify resolution policies
- negotiate a compromise
- tolerate inconsistency
- combine the above techniques

Topics



Log of Interactions and Resolutions

Event: routeselected

World: 18 Interaction Occured.

(

(Ipsf-rule-one 1 Steve Marsha Marsha)

(true))

Conflicting Action: (Ipsb-rule-one 1 Steve Marsha Marsha)

Event: routeselected

World: 42 Interaction Occured.

(

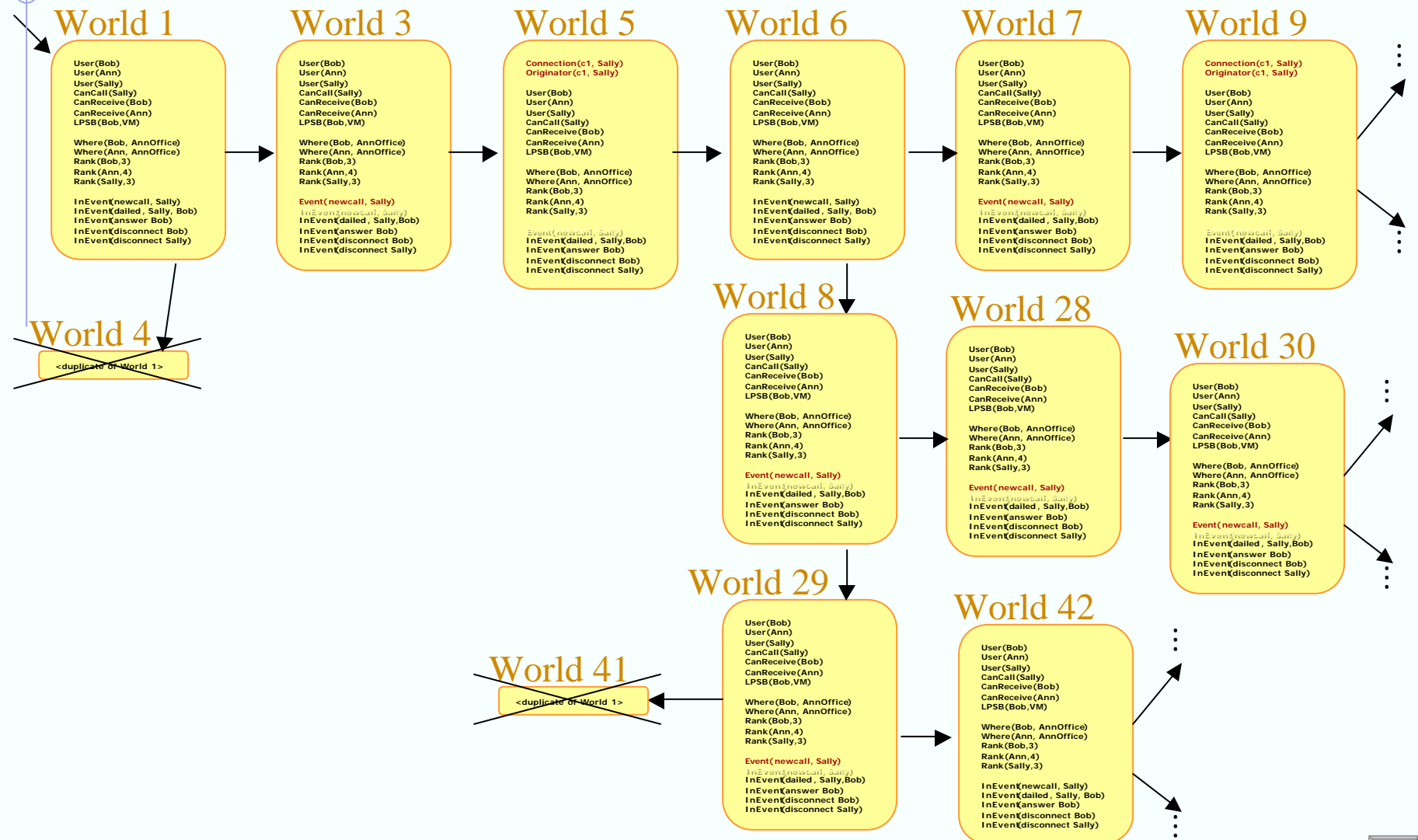
(Ipsf-rule-one 1 Steve Marsha Marsha)

(true))

Conflicting Action: (Ipsb-rule-one 1 Steve Marsha Marsha)



Reachable (conflict-free) State Space



Demo

Boss in Room

If a higher ranking person is in room, forward to voice mail calls destined for me

if: Call(?c, ?t, allocate, ?o, Marsha)

and: PhoneInRoom(?t, ?t_room)

and: Where(Marsha, ?t_room) **and** Where(?p, ?t_room)

and: Rank(Marsha) < Rank(?p) **and** Rank(?o) < Rank(?p)

then: –Connection(?c, ?t)

then: Message(Redirect, ?c, ?o, Marsha, MarshaVM)

Call Forward to Location

Forward my calls to my current location

if: Call(?c, ?t, allocate, ?o, Marsha)

and: Where(Marsha, ?p_room) **and** Owner(?p, ?p_room)

then: –Connection(?c, t)

then: Message(Redirect, ?c, ?o, Marsha, ?p)

Experiments

We are experimenting with different resolution strategies, to identify those that are generally effective, computationally feasible, and produce predictable behaviours.

Features:

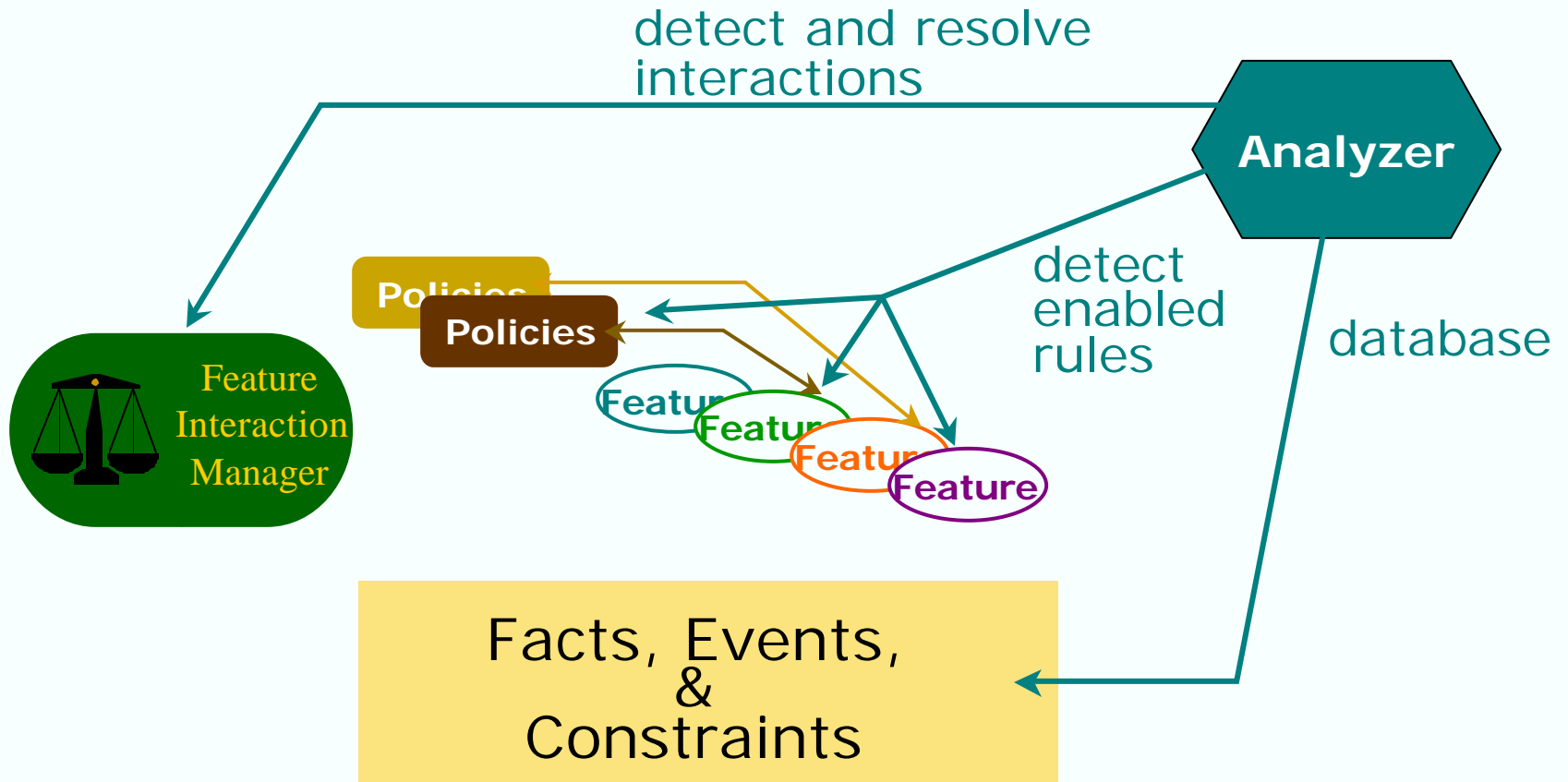
- POTS
- Call Screenings
- Call Forward (U, B, NA, VM)
- Call Waiting
- Three-Way Calling
- Distinct Ringing
- Automatic Recall
- Call Return
- “Boss in Room”
- “Call Forward to my Location”
- “Forward Person/Group to Secretary”
- “Notify me when Person is in Location”
- “Notify me when Team Members Gather”

Resolution Strategies:







- Feature priority
- Action priority

Run-Time System

We're in the process of distributing the analyzer functionality to run-time agents.



Summary

-  Rapid, modular, dynamic development of features
-  Predictable behaviour of feature combinations
-  Simple, expressive feature modelling language
-  Static analyzer that reports conflict resolutions
-  Experiments to evaluate resolution strategies
-  Run-time system to resolve interactions

Policies

Information that modifies system behaviour

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

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

- Forward calls to voice mail if boss is in the room
- Notify me when Ann is in her office