

From System Goals to Software Architecture

Axel van Lamsweerde

University of Louvain
B-1348 Louvain-la-Neuve (Belgium)

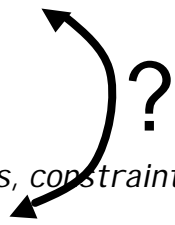
SFM-03: Software Architecture
Bertinoro, 22/09/03

Two essential activities in the SE process ...

- ◆ Requirements Engineering (RE) =
elicit, specify, analyze & document ...
objectives, functionalities, qualities, constraints
⇒ structured models of *system-to-be*
- ◆ Architectural Design (AD) =
organize, specify, analyze & document ...
components, interactions, configurations, constraints
⇒ structured model of *software-to-be*

Architecture has big impact on achieving NFRs

The problem ...

- ◆ Requirements Engineering (RE) =
elicit, specify, analyze & document ...
objectives, functionalities, qualities, constraints
⇒ structured models of *system-to-be*
 - ◆ Architectural Design (AD) =
organize, specify, analyze & document ...
components, interactions, configurations, constraints
⇒ structured model of *software-to-be*
- Architecture has big impact on achieving NFRs*
- 

The problem ... (2)

- ◆ Poor understanding of...
 - relationships *requirements ↔ architecture*
 - intertwining *RE ↔ AD*
 - ◆ No systematic way to ...
 - build/modify architecture to meet functional/non-functional requirements
 - integrate architectural constraints in requirements document
- ⇒ *requirement-architecture mismatch*

The mismatch problem: exacerbating factors ...

- ◆ Requirements volatility vs. architectural stability
(e.g. new requirements from using the software)
- ◆ New generation software ...
 - ubiquitous, mobile
 - heterogeneous
 - open
 - mission-critical
 - operating in changing, (hostile) environments
 - open source (permanent, distributed evolution)

Resolving the mismatch problem:
why not just forget about requirements ??

- ◆ Survey of 350 US companies, 8000 projects
 - success: 16 %
 - failure: 33 %
 - so so: 51 %

**(partial functionalities,
excessive costs, big delays)**

major source of failure:

poor requirements engineering @ 50% responses

(Standish Group, 1995)

Resolving the mismatch problem:
why not just forget about requirements ??

Major source of failure:

poor requirements engineering @ 50% responses:

- lack of user involvement 13%
- incomplete requirements 13%
- changing requirements 9%
- unrealistic expectations 10%
- unclear goals 5%

www.standishgroup.com/chaos.html

Resolving the mismatch problem:
why not just forget about requirements ??

◆ Survey of 3800 EUR organizations, 17 countries

main software problems are in...

- requirements specification
> 50% responses
- requirements management
50% responses

(European Software Institute, 1996)



The problem on the research side ...

- ◆ Much work on architectural description & analysis
 - myriads of ADLs:
ACME, C2, DARWIN, RAPIDE, WRIGHT, UML2.0 (?), ...
the architecture has to be there
 - architectural patterns & styles
how do you compose them to meet NFRs ?
- ◆ Some work on architectural refinement
 - e.g., [Moriconi'96]



The problem: on the research side ... (2)

- ◆ Little work on architecture derivation *to meet* functional & non-functional reqs
 - some preliminary efforts on goal-oriented approaches for...*
 - iterative evaluation/transformation against NFRs [Bosch&Molin '99]
 - architectural refinement [van Lamsweerde'00]
 - NFR-based documentation of design patterns for selection [Gross&Yu'01]

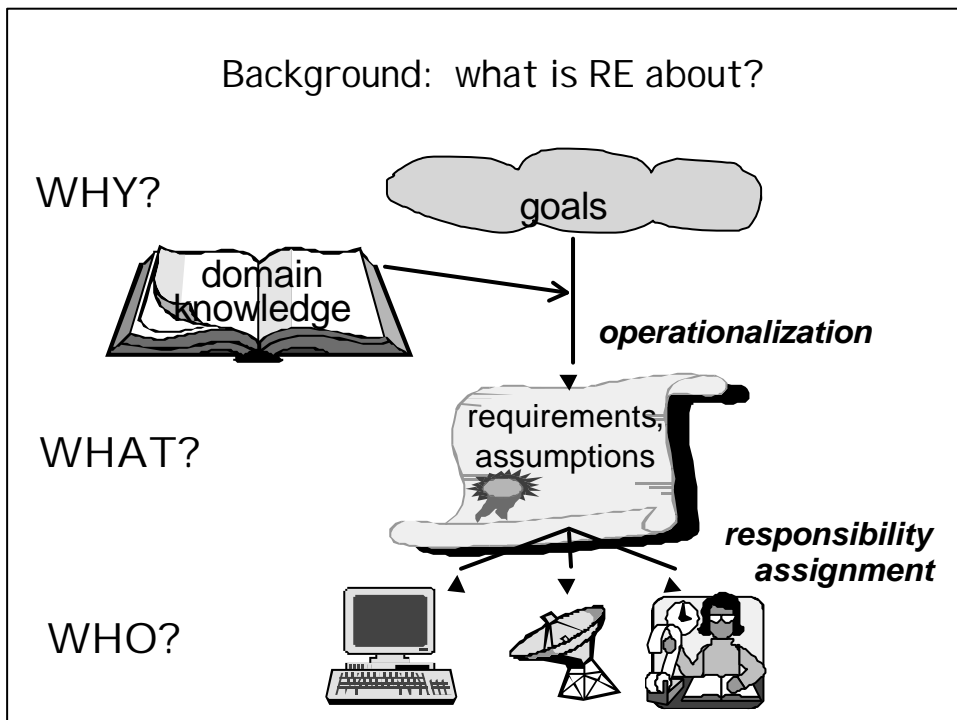
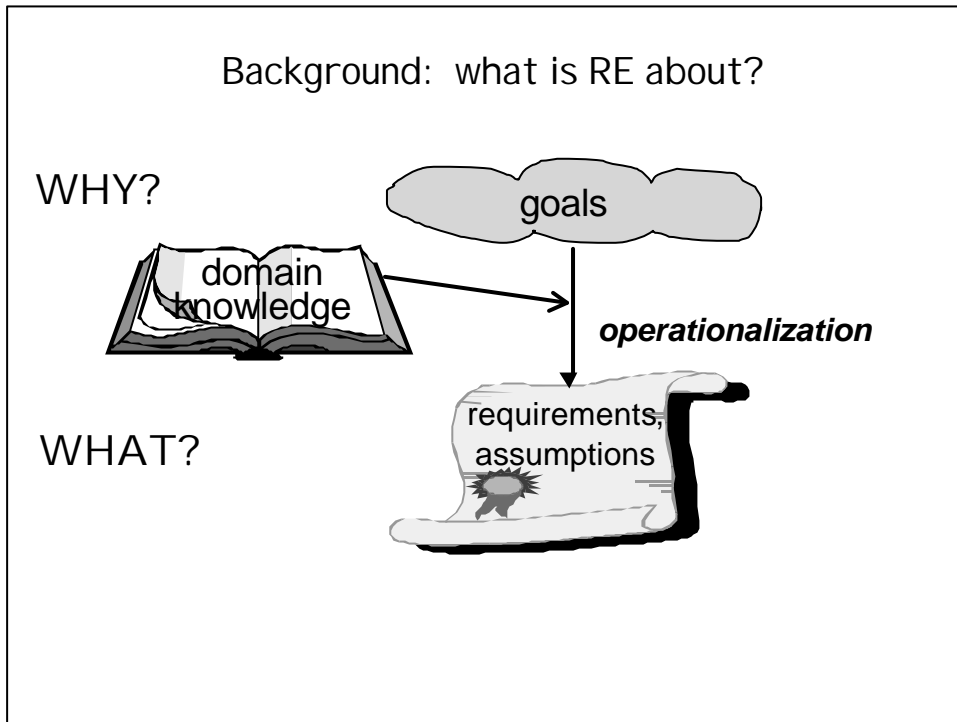


Objectives

- ◆ Support requirements/architecture co-design/co-evolution
 - ◆ Support architecture derivation from requirements models & software specs
 - ◆ Make derivation process...
 - systematic, incremental
 - leading to provably/arguably correct & "good" architecture
 - highlighting architectural views (e.g. security view)
- β
- goal-based architectural design process

Outline

- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-level reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve functional specs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs



Background: what is RE about?

- ◆ Requirements elaboration is hard ...
 - requirements are not there,
you have to elicit them & structure them
 - ranges from high-level, strategic objectives
to detailed, technical requirements
 - involves software + environment
 - requires evaluation of alternatives, selection
(= *architectural decisions* ?)
 - raises conflicting concerns
 - requires anticipation of unexpected behaviors
(for requirements completeness, system robustness)

Background: goal-oriented RE

- ◆ Goal: prescriptive statement of intent
(cf. David 's notion of intention/task)
- ◆ Domain prop: descriptive statement about domain
- ◆ Agent: active component, controls behaviors
software-to-be, existing software, device, human
Goal achievement requires agent cooperation
The more fine-grained a goal is, the less agents are required
- ◆ Requirement: goal assigned to software agent
- ◆ Expectation: goal assigned to environment agent

Background: goal-oriented RE (2)

Different goal categories ...

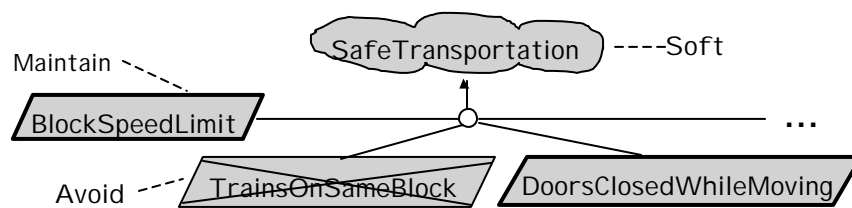
- ◆ functional: prescribe expected services
satisfaction, information, ...
- ◆ non functional, refined in application-specific terms:
 - quality of service:
accuracy
security: confidentiality, availability, integrity, ...
usability
performance, ...
 - development goals:
maintainability: min coupling, max cohesion, ...
reusability, interoperability, ...
 - domain-specific architectural constraints

Background: goal-oriented RE (3)

- ◆ Domain-specific architectural constraints ...
 - features of environment agents & their organization
 - constrain architectural design space
- e.g. distribution of human agents, devices, data
- Meeting scheduling system:
distribution of participants, meeting initiator
- Train system:
station computer, on-board controller,
tracking system, ...

Background: goal-oriented RE (4)

- ◆ Different types of goals ...
 - SoftGoal achievement cannot be established in clear-cut sense
 - Ⓜ goal satisficing, qualitative reasoning
(Mylopoulos'92, Chung'00)
 - Achieve/Maintain goal achievement can be verified
 - Ⓜ goal satisfaction, formal reasoning
(Dardenne'93, Darimont'96)



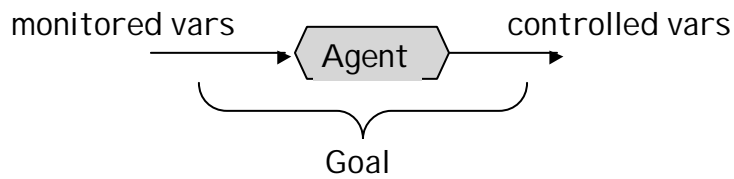
Background: goal-oriented RE (5)

- ◆ Goal G is AND-refined into subgoals G_1, \dots, G_n iff achieving G_1, \dots, G_n contributes to achieving G
 - the set $\{G_1, \dots, G_n\}$ is called refinement of G
 - G_i is said to contribute positively to G
- ◆ The set $\{G_1, \dots, G_n\}$ is a complete AND-refinement of G iff G_1, \dots, G_n are sufficient for achieving G in view of known domain properties

$$\{G_1, \dots, G_n, \text{Dom}\} \models G$$
- ◆ Goal G is OR-refined into refinements R_1, \dots, R_m iff achieving the subgoals of R_i is one alternative to achieving G ($1 \leq i \leq m$)
 - R_i is called alternative for G

Background: goal-oriented RE (6)

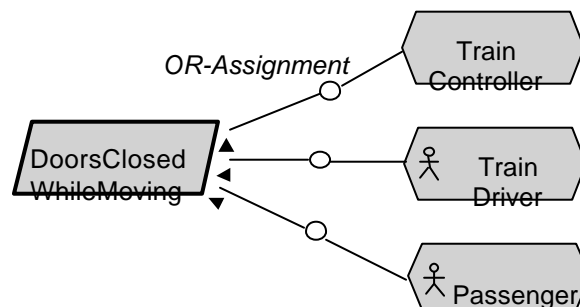
- ◆ A goal is realizable by agent if it amounts to a relation on variables that are monitorable & controllable by the agent

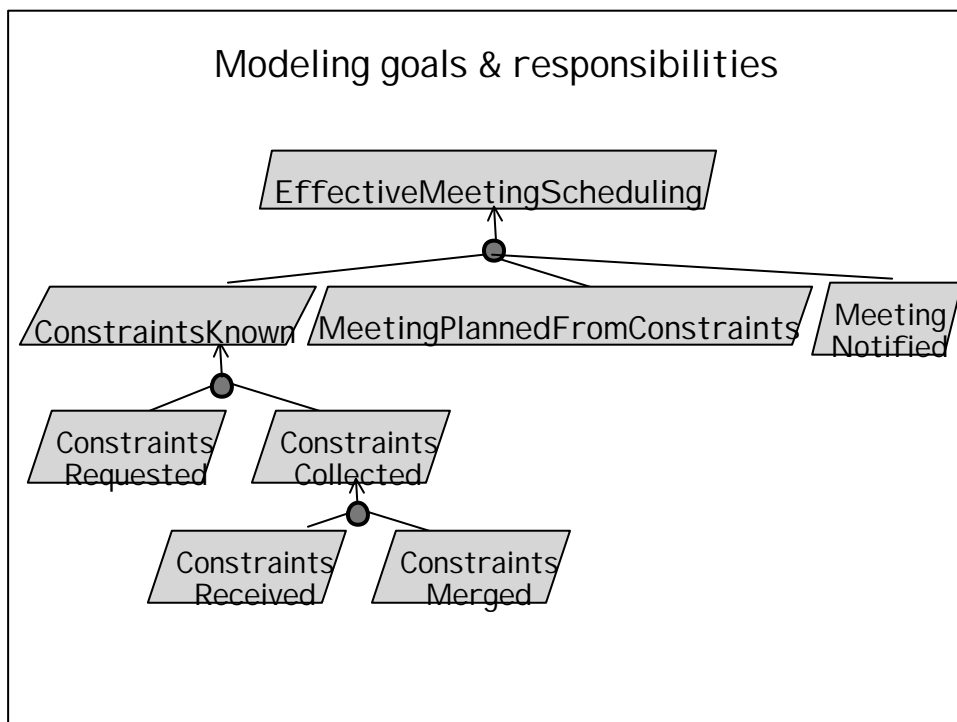
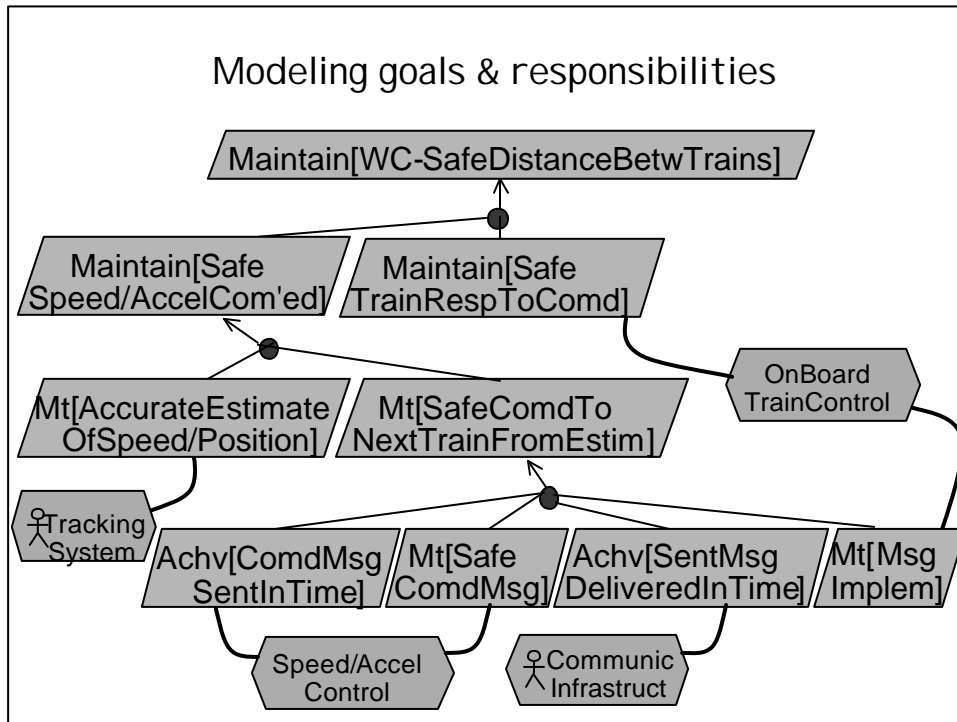


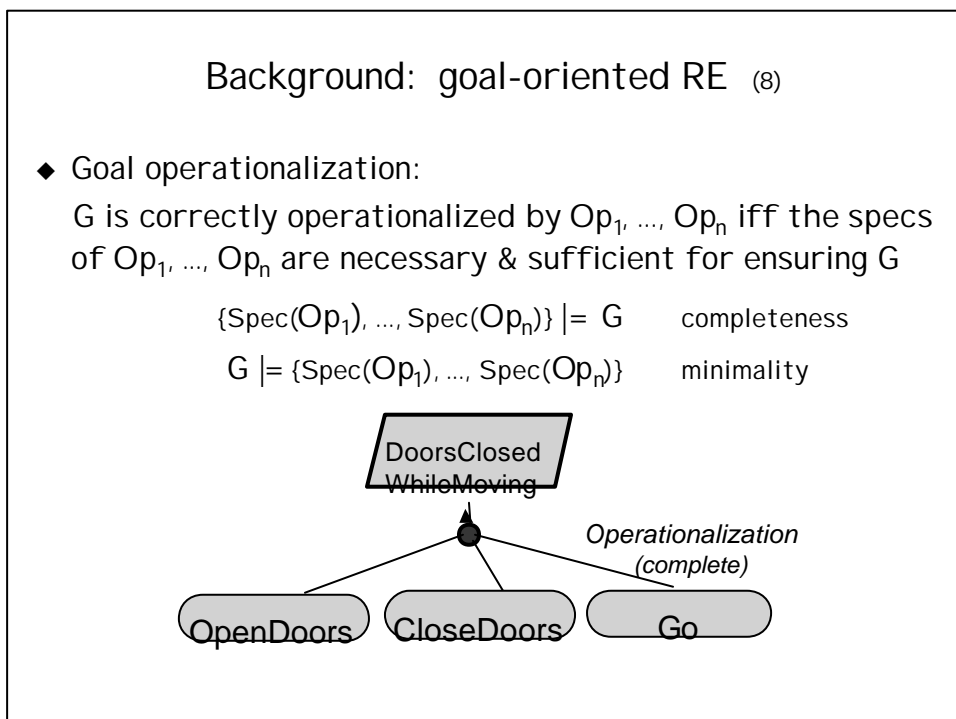
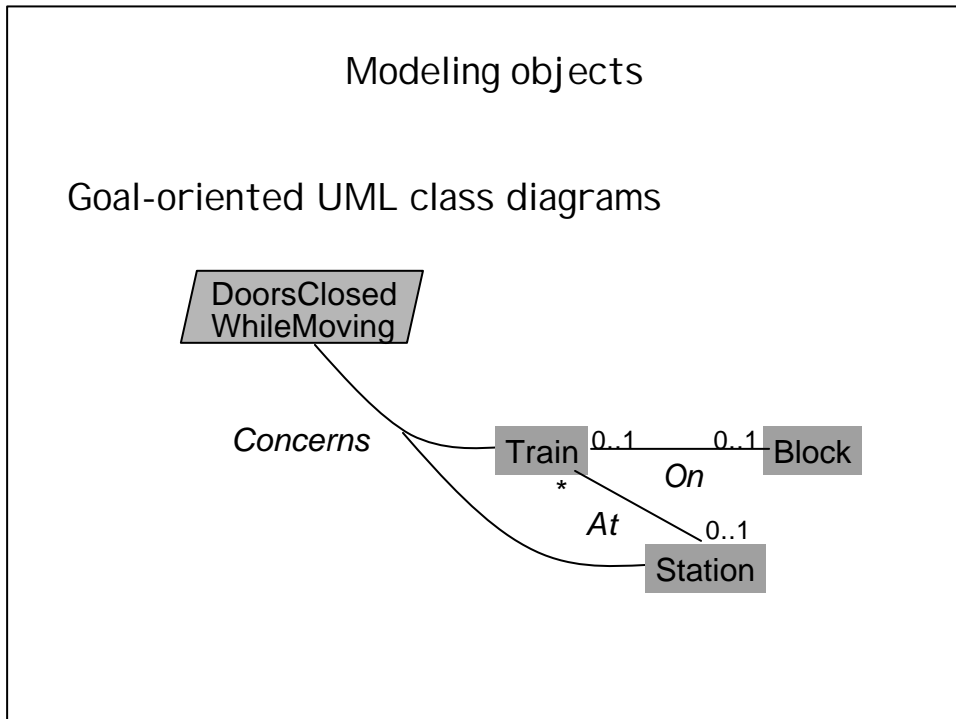
Goals need to be refined until assignable to single agents

Background: goal-oriented RE (7)

- ◆ Agent responsibility:
G is assignable to Ag iff G is *realizable* by Ag





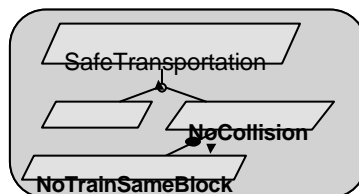


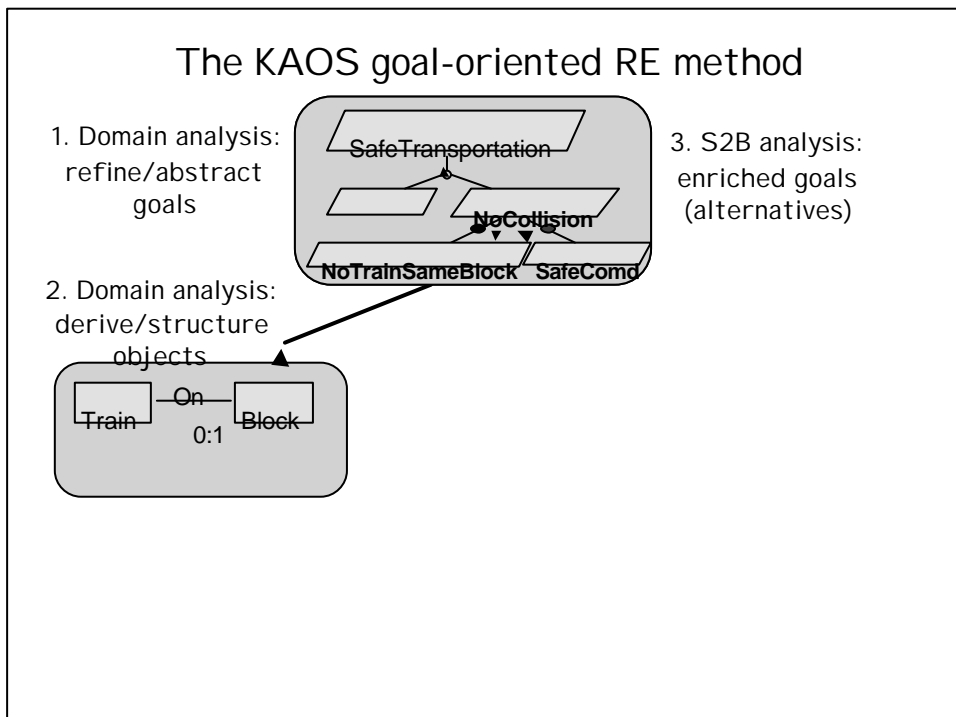
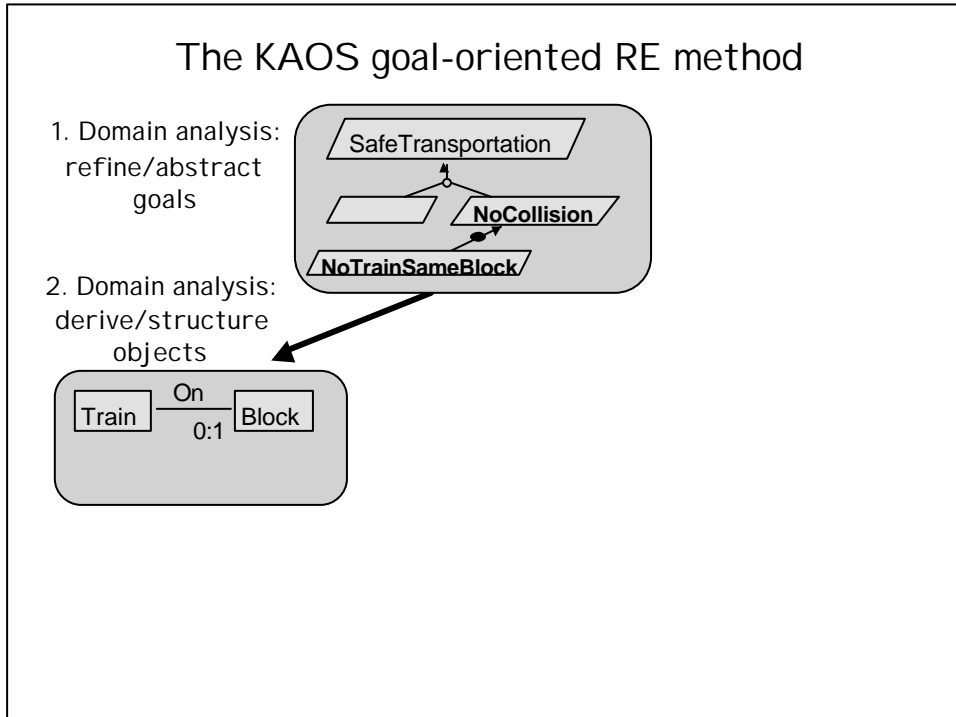
Outline

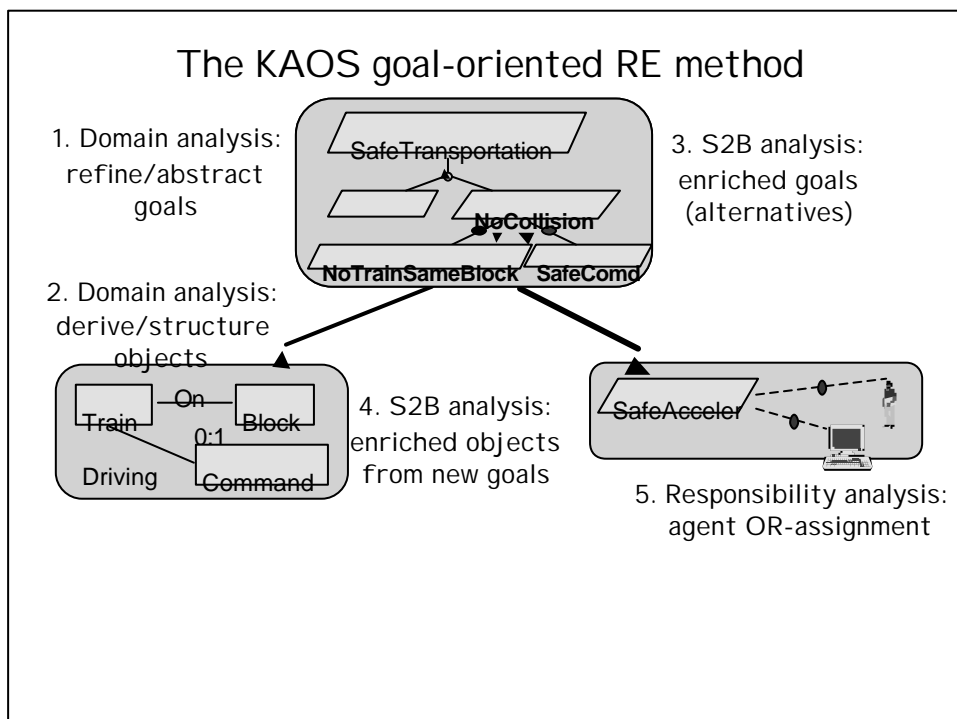
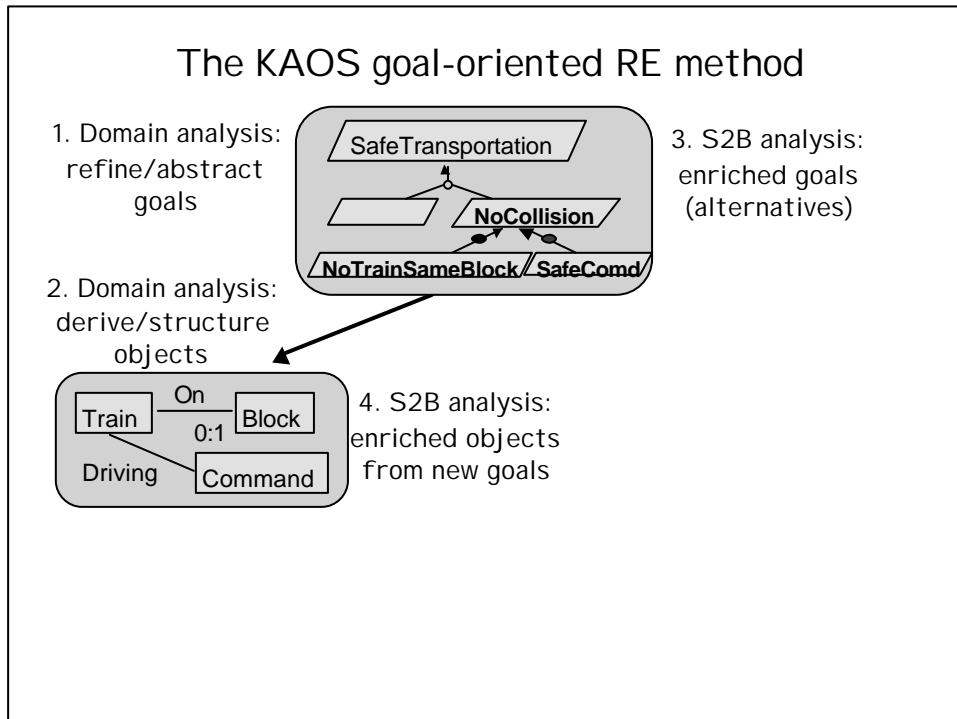
- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-based reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve FRs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs

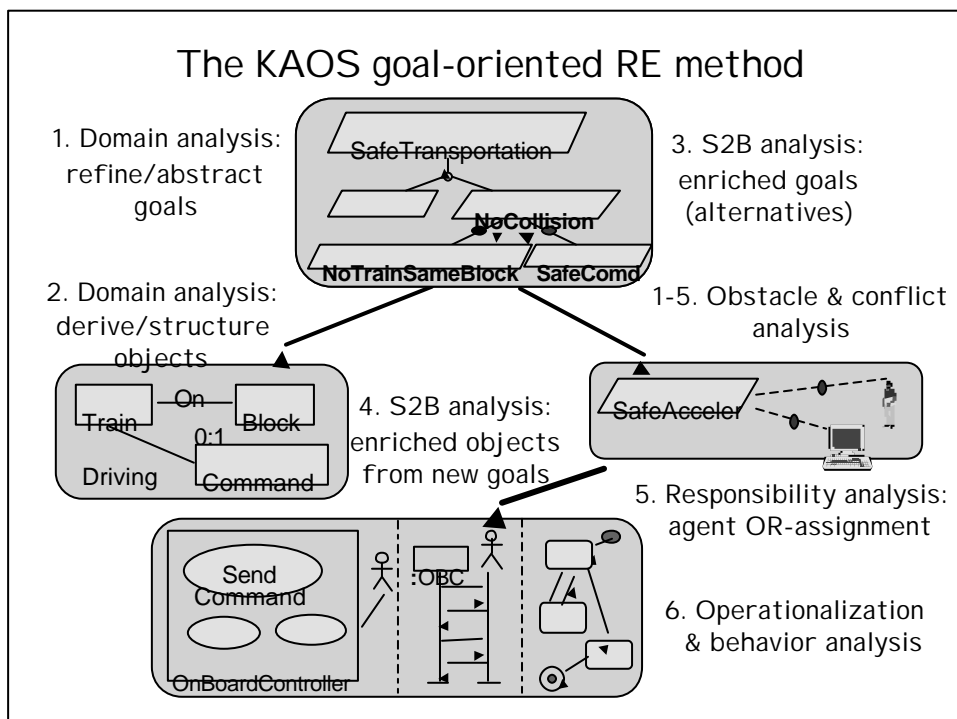
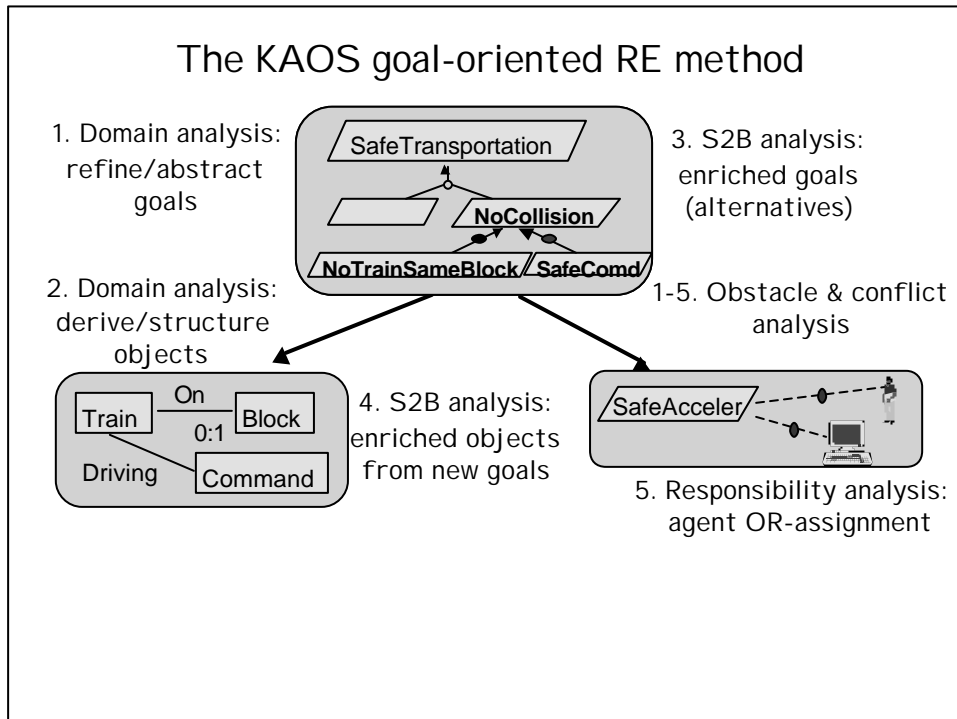
The KAOS goal-oriented RE method

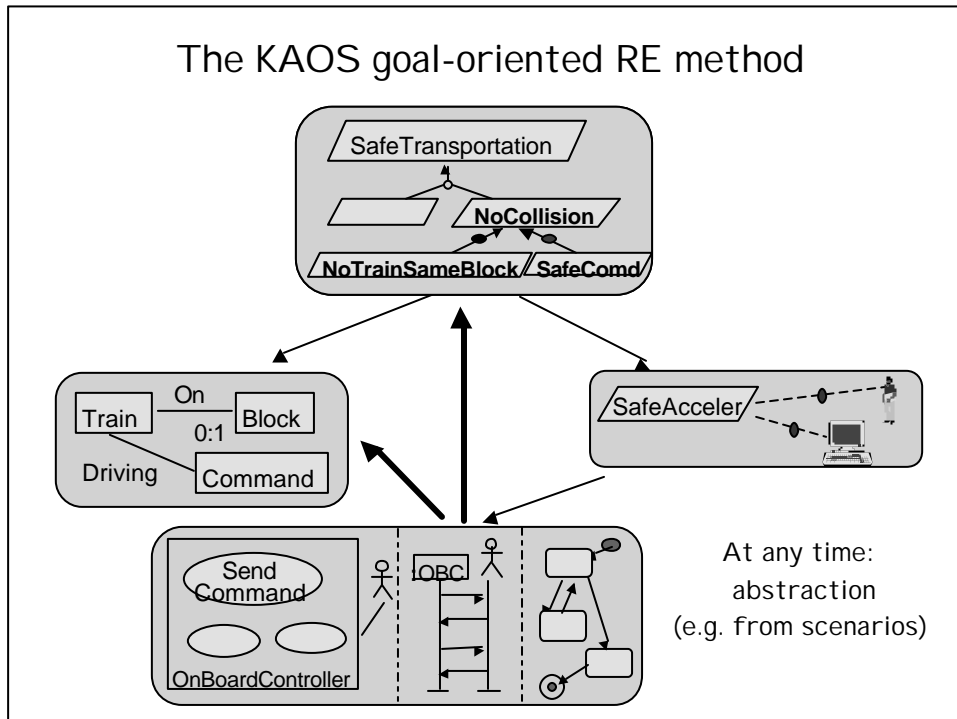
1. Domain analysis:
refine/abstract
goals











Specifying goals, objects & operations

Formal specification is optional ...

- to support more formal analysis & derivations
- in KAOS:
 - only when & where needed
 - abstract language for goals, requirements, assumptions, domain properties:
 - real-time temporal logic
 - more operational language for operations:
 - state-based spec
 - with *traceability to underlying goals*

Some bits of real-time temporal logic

- o P: P shall hold in the next state
- P: P shall hold in every future state
- P **W** N: P shall hold in every future state unless N holds
- à P: P shall hold in some future state
- _{≤T} P: P shall hold in every future state up to T time units
- à_{≤T} P: P shall hold within T time units
- + past operators: "black" symbols
- @P: • ¬ P ∪ P

Specifying goals: formal

Goal *Maintain* [DoorsClosedWhileMoving]

...

FormalDef $\forall tr: Train, s: Station$

$At(tr, st) \hat{\cup} \bullet \neg At(tr, st) \Rightarrow$

$tr.Door = "closed" \mathbf{W} At(tr, next(st))$

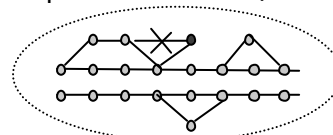
Goal *Achieve* [NoDelay]

...

FormalDef $\forall tr: Train, s: Station$

$At(tr, st) \Rightarrow \hat{a}_{\leq T} At(tr, next(st))$

*characterizes maximal set of
intended behaviors*



Specifying operations: formal

Operation OpenDoors

Input tr: Train ; Output tr': Train

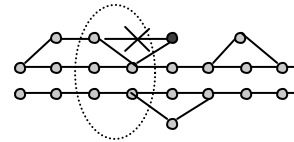
DomPre tr.Doors = "closed" domain description

DomPost tr.Doors = "open"

ReqPre for *DoorsClosedWhileMoving*: permission
 $\exists s$: Station At (tr, s)

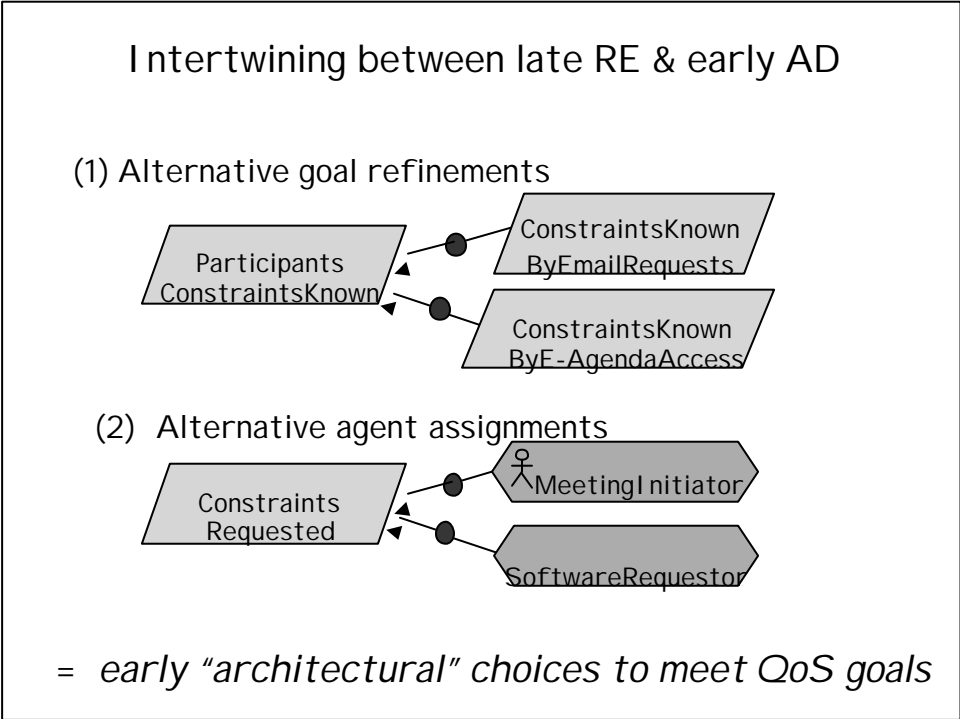
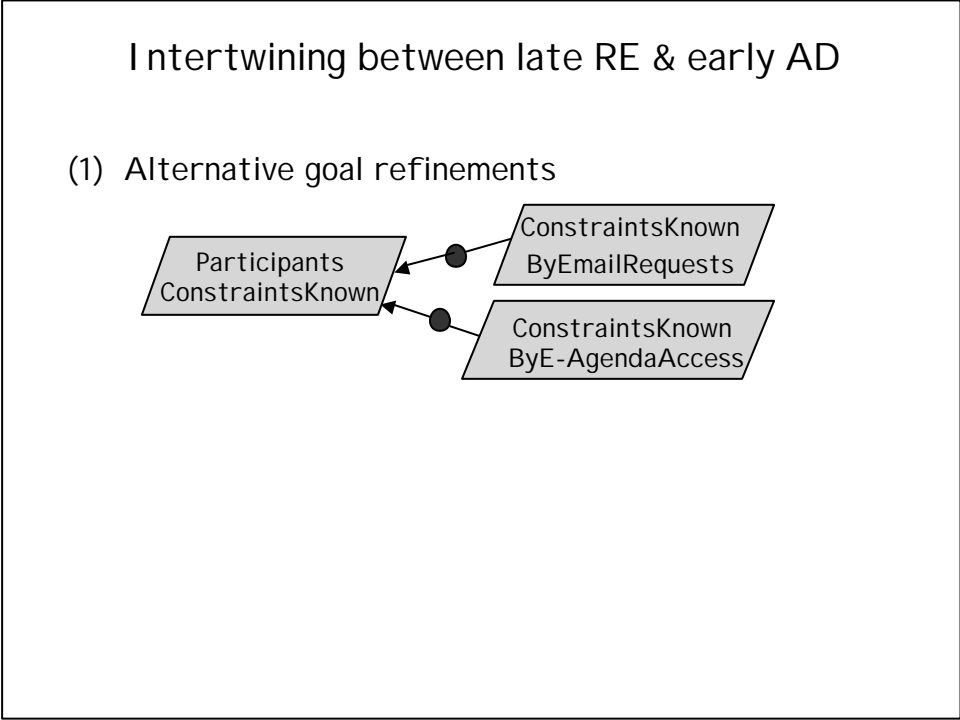
ReqTrig for *NoDelay*: obligation
Stopped (tr)

*characterizes maximal set of
intended states at snapshot*



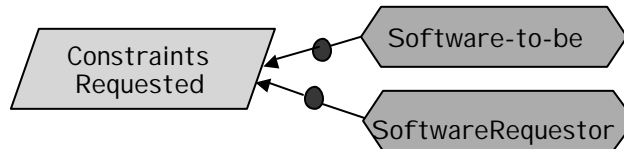
Outline

- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-level reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve functional specs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs



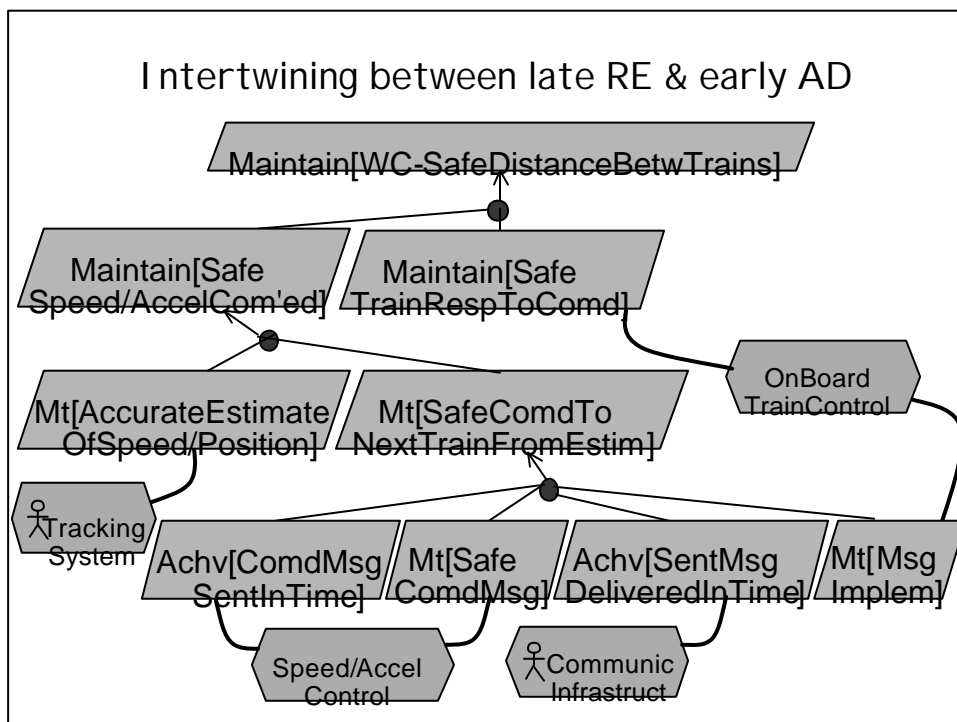
Intertwining between late RE & early AD

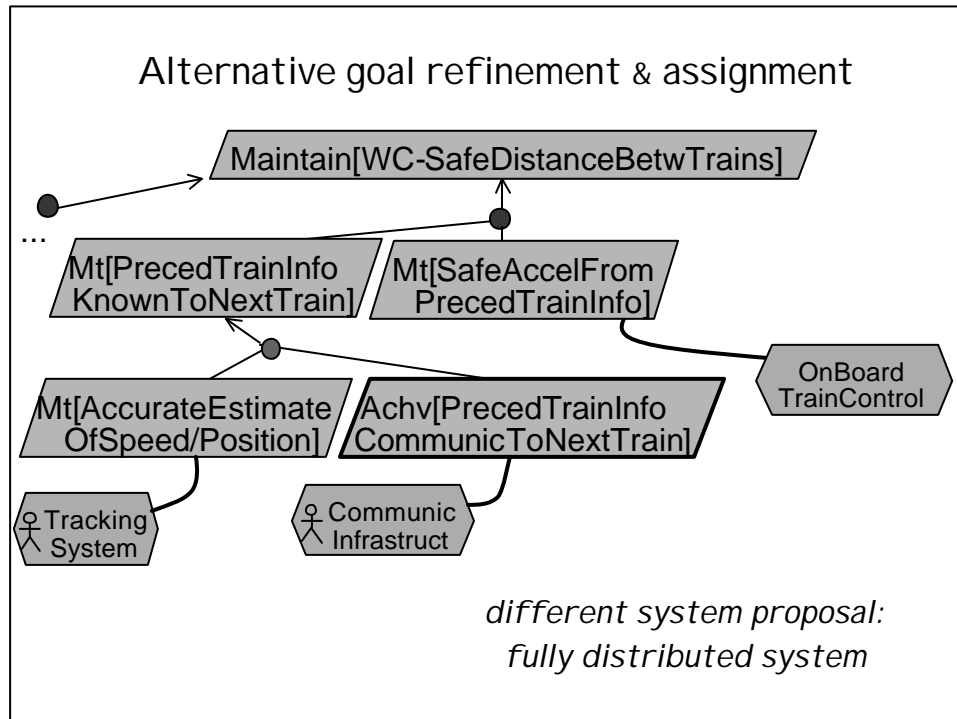
(3) Alternative granularities for software agents



Fine, function-level granularity will be selected to meet NFR Maximize [Cohesion (C)]

Intertwining between late RE & early AD





- Outline
- ◆ Background: some bits of RE
 - ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-level reasoning for higher assurance
 - ◆ From software requirements to software specs
 - ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve functional specs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs

Formal goal-level reasoning for higher assurance

- ◆ Early analysis on partial models, intertwined with model construction

Wide range of opportunities:

- checking/deriving goal refinements
- checking/deriving operationalizations
- generating obstacles
- generating boundary conditions for conflict
- goal mining from scenarios
- generating state machines from operationalizations
- reusing goal-based specs by analogy

Formal goal-level reasoning for higher assurance

- ◆ Early analysis on partial models, intertwined with model construction

◆ Wide range of opportunities:

- checking/deriving goal refinements
- checking/deriving operationalizations
- generating obstacles
- generating boundary conditions for conflict
- goal mining from scenarios
- generating state machines from operationalizations
- reusing goal-based specs by analogy

Checking goal refinements

- ◆ Aim: show that refinement is correct & complete
R, Ass, Dom \vdash G
R: conjunctive set of requirements or subgoals

Checking goal refinements

- ◆ Aim: show that refinement is correct & complete
R, Ass, Dom \vdash G
R: conjunctive set of requirements or subgoals
- ◆ Approach 1: use TL theorem prover
heavyweight, non-constructive

Checking goal refinements

- ◆ Aim: show that refinement is correct & complete
 $R, Ass, Dom \models G$
 R: conjunctive set of requirements or subgoals
- ◆ Approach 1: use TL theorem prover
 heavyweight, non-constructive
- ◆ Approach 2: use formal refinement patterns
 lightweight, constructive:
 - to complete partial refinements
 - to explore alternative refinements

Checking goal refinements (2)

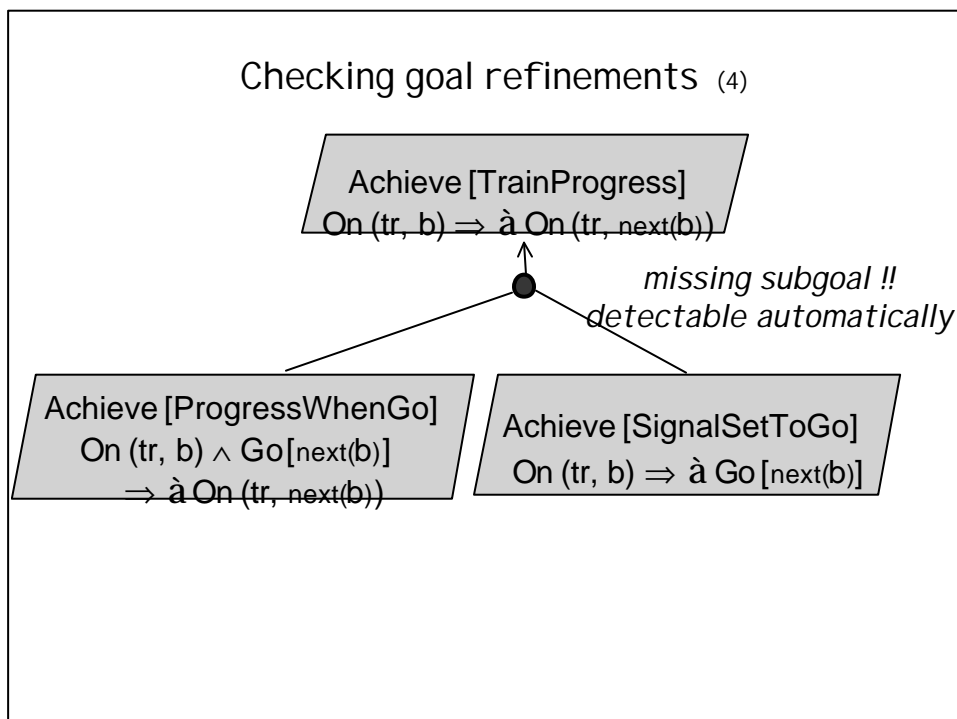
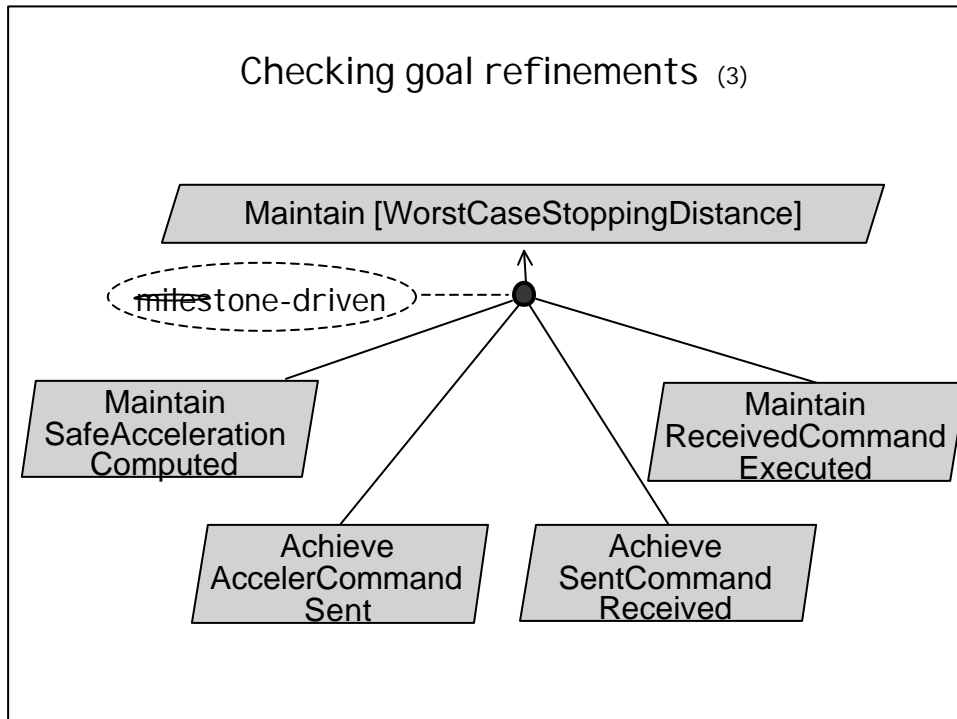
Idea:

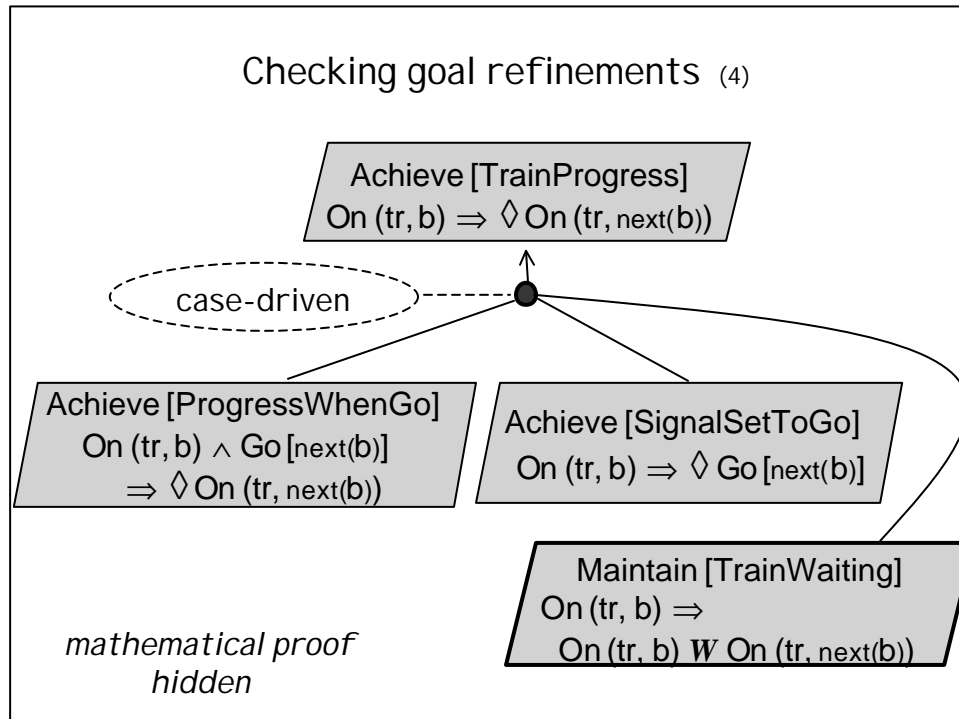
- ◆ Build library of patterns (structured by *tactics*)
- ◆ Prove patterns once for all
- ◆ Reuse through instantiation, in matching situation

e.g. frequent patterns:

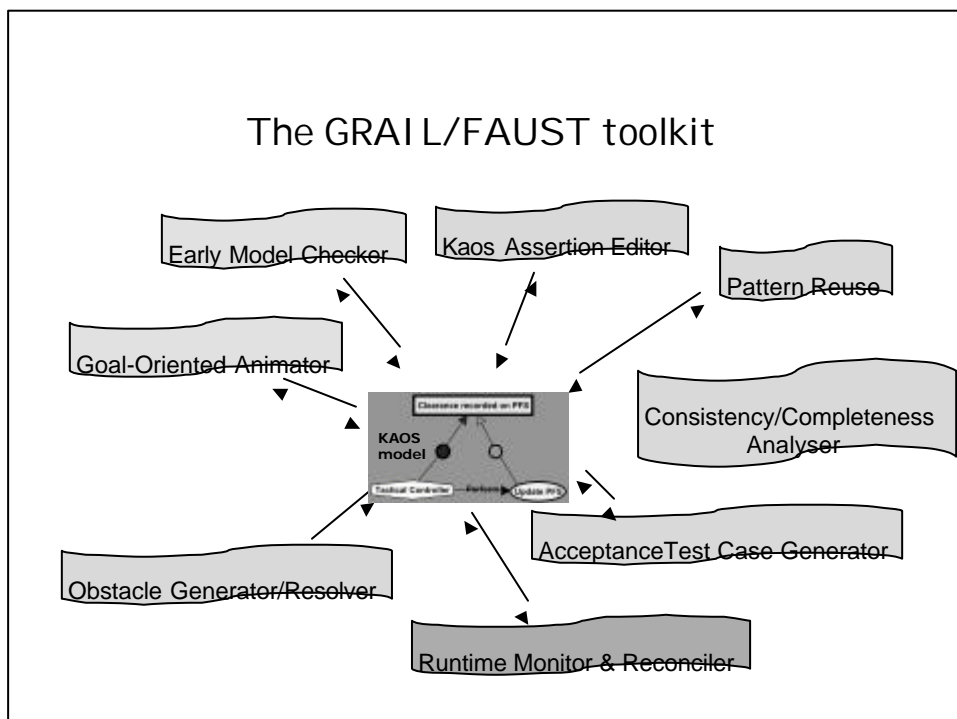
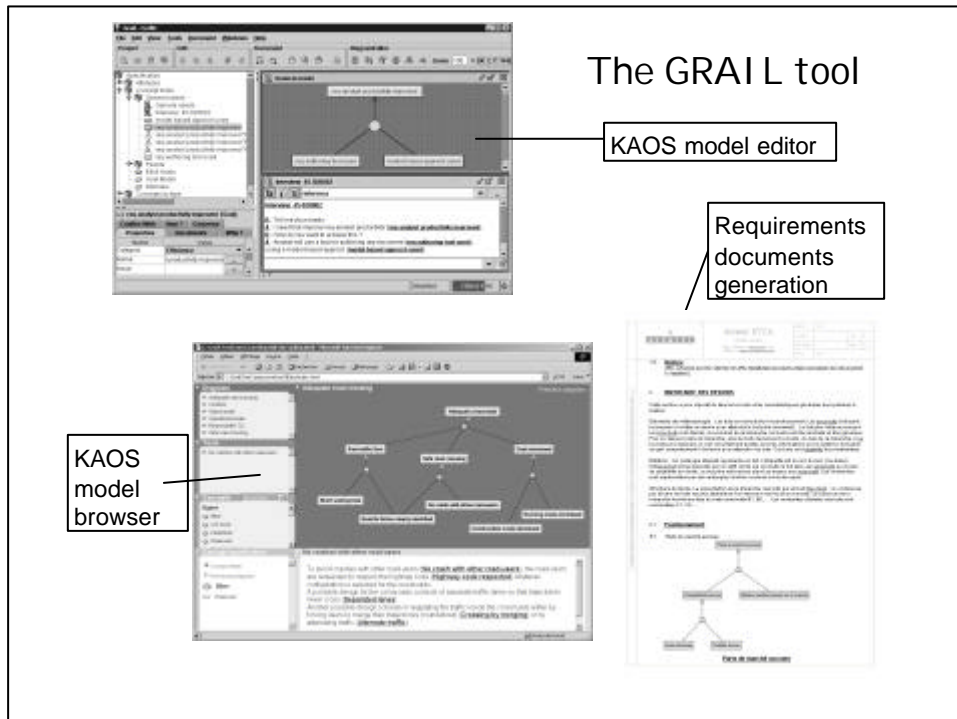
milestone-driven

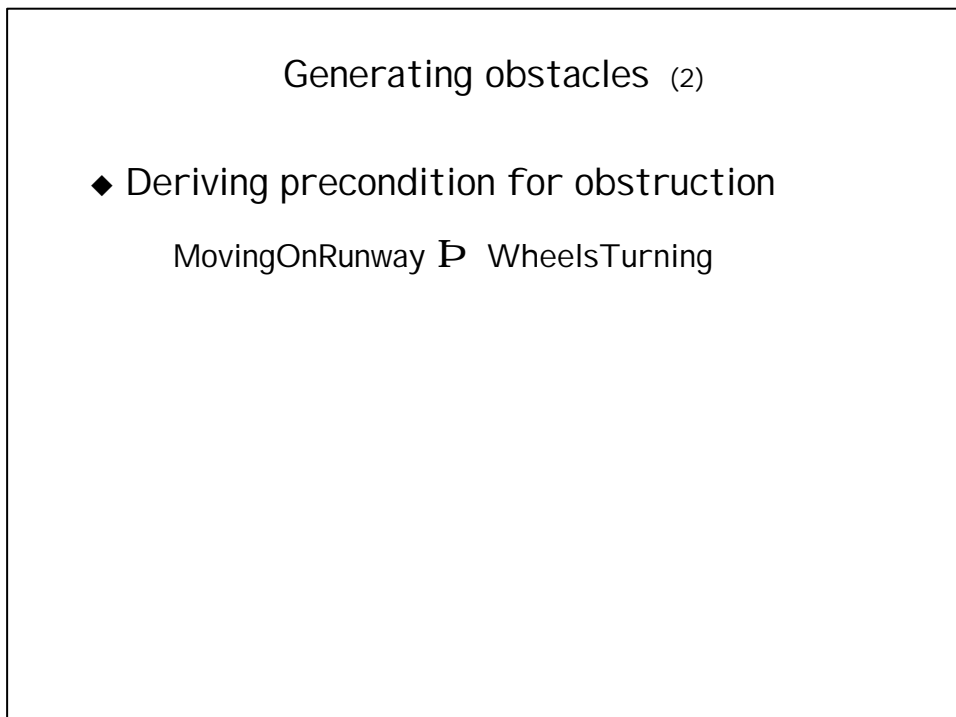
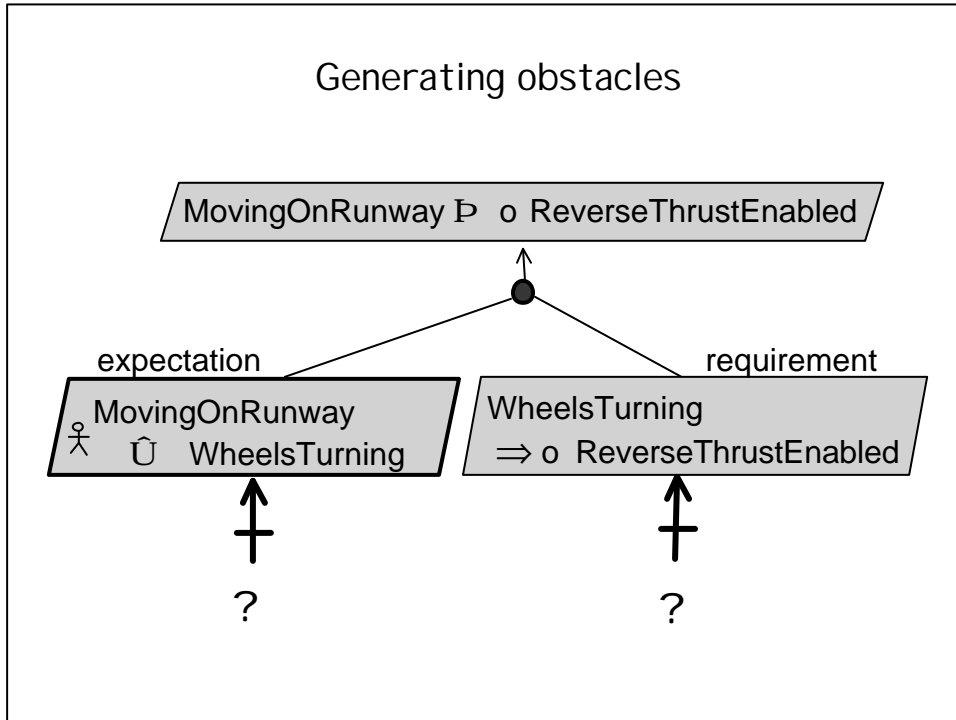
case-driven





- Checking goal refinements (5)
- ◆ Approach 3: Early bounded model checking
 - checking of goal models
 - partial models
 - incremental checking/debugging
 - on selected object instances (*propositionalization*)
 - output:
 - OK
 - KO + counter-example scenario
- Roundtrip use of SAT solver, NuSMV, theorem prover*
- Time for demo...





Generating obstacles (2)

◆ Deriving precondition for obstruction

MovingOnRunway \mathcal{P} WheelsTurning

Ⓜ goal negation:

à MovingOnRunway $\hat{\mathcal{U}}$ \neg WheelsTurning

Generating obstacles (2)

◆ Deriving precondition for obstruction

MovingOnRunway \mathcal{P} WheelsTurning

Ⓜ goal negation:

à MovingOnRunway $\hat{\mathcal{U}}$ \neg WheelsTurning

Ⓜ regress through Dom:

? necessary conditions for wheels turning ?

WheelsTurning \mathcal{P} \neg Aquaplaning

i.e. Aquaplaning \mathcal{P} \neg WheelsTurning

Generating obstacles (2)

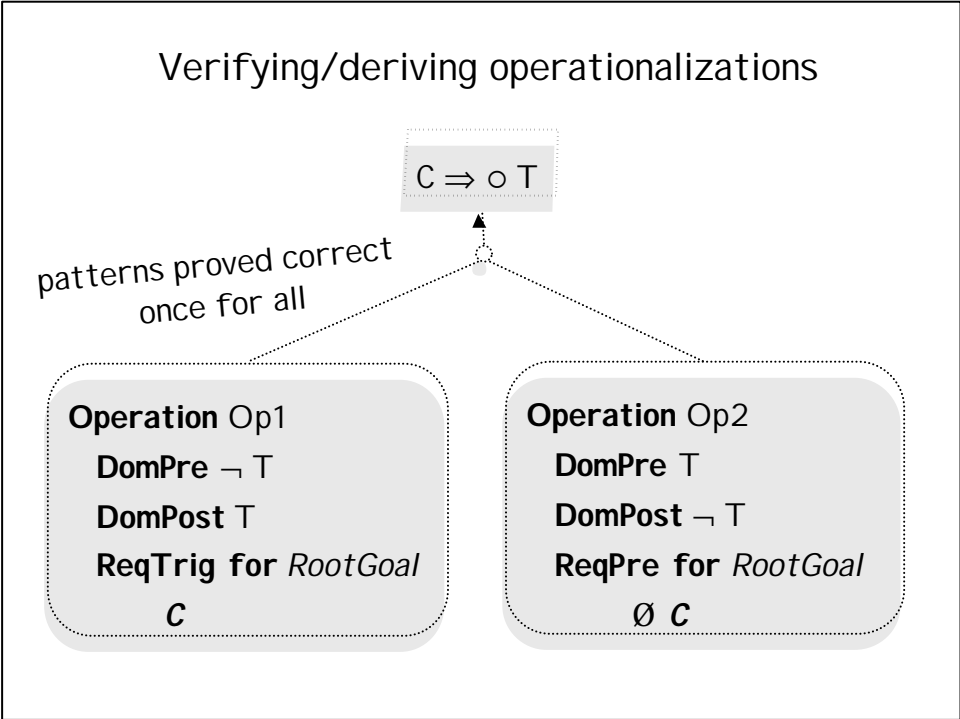
- ◆ Deriving precondition for obstruction
 - MovingOnRunway \mathcal{P} WheelsTurning
 - Ⓜ goal negation:
 - à MovingOnRunway \hat{U} \neg WheelsTurning
 - Ⓜ regress through Dom:
 - ? necessary conditions for wheels turning ?
 - WheelsTurning \mathcal{P} \neg Aquaplaning
 - i.e. Aquaplaning \mathcal{P} \neg WheelsTurning
 - Ⓜ RHS unifiable:
 - à MovingOnRunway \hat{U} Aquaplaning *Warsaw obstacle*

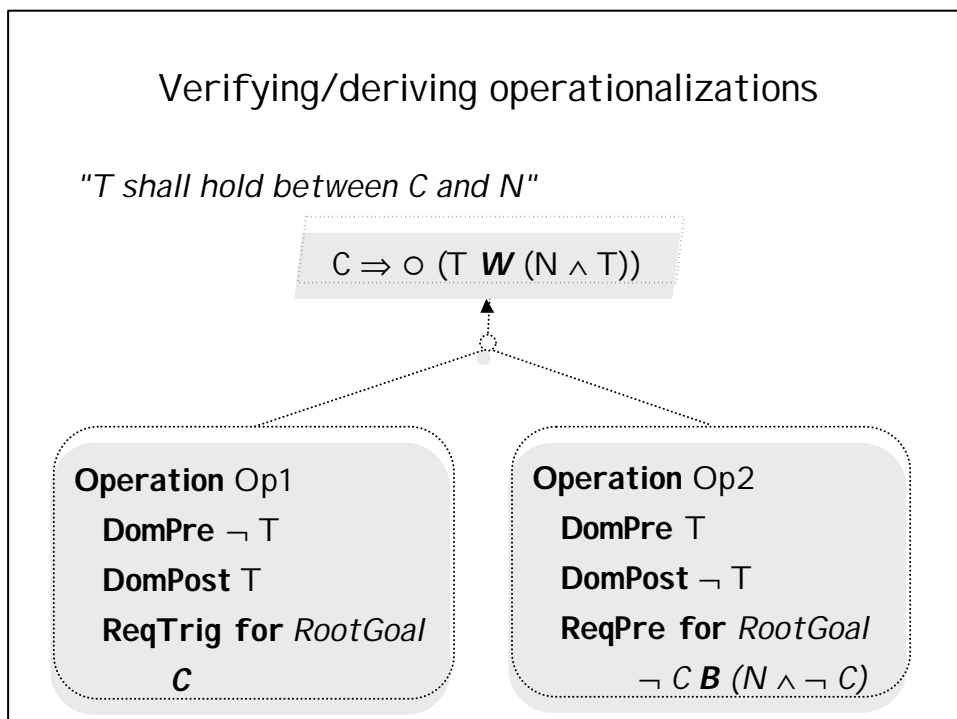
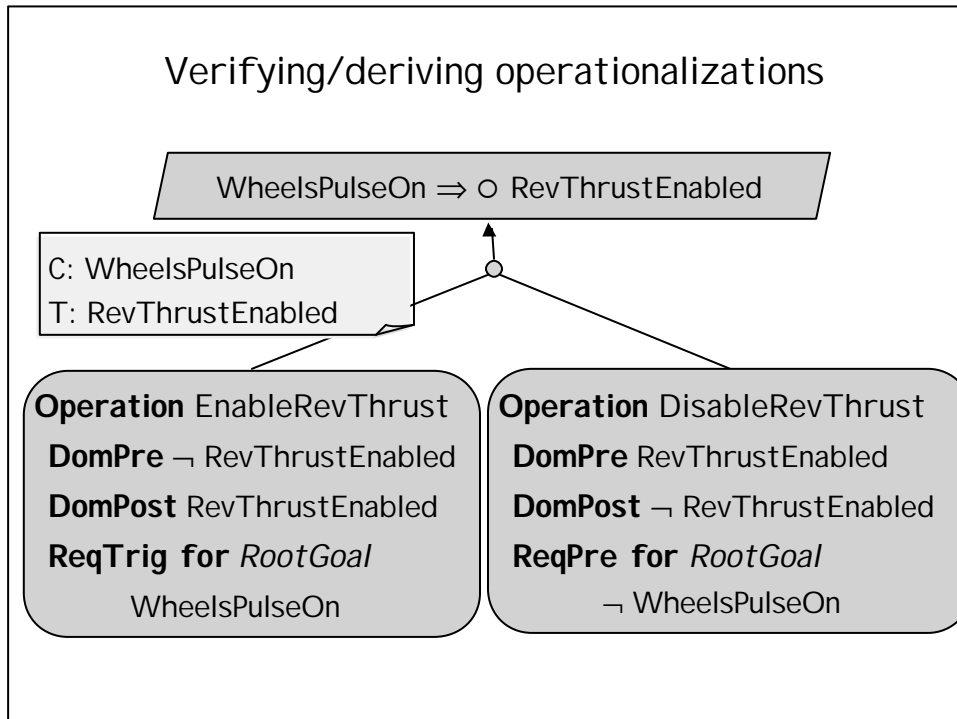
Generating obstacles (3)

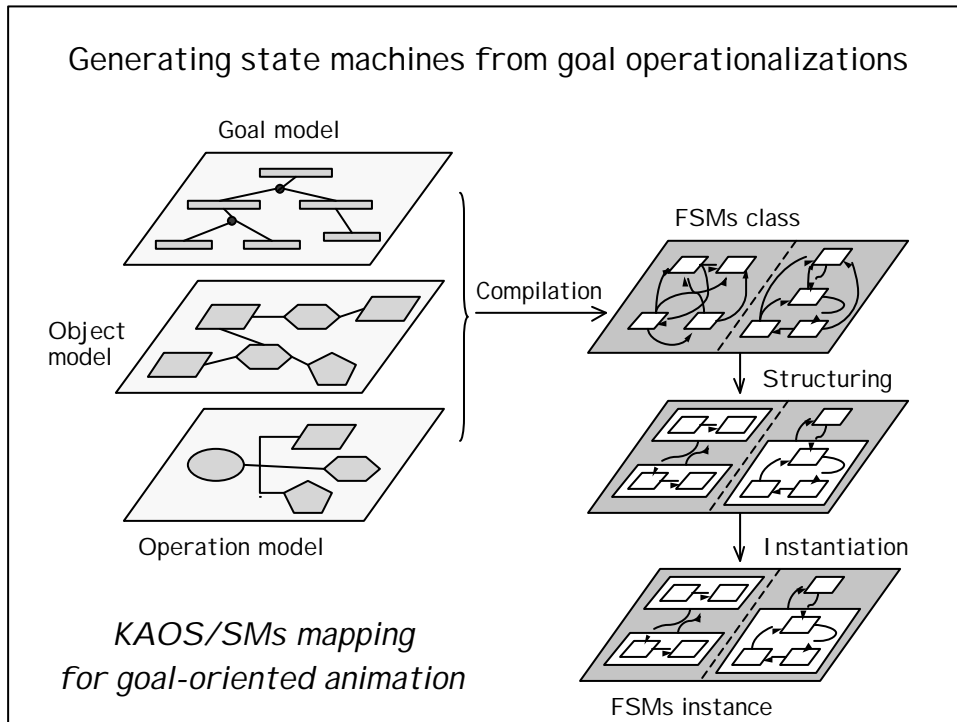
- ◆ Using formal obstruction patterns
 - in fact we just used a frequent pattern:*

Verifying/deriving operationalizations

- ◆ Build a library of formal operationalization patterns for frequent goal specification patterns
 - e.g. Achieve goals: $C \Rightarrow \mathring{a}_{\leq d} T$ $C \Rightarrow \circ T$
 - Maintain goals: $C \Rightarrow \square T$ $C \Rightarrow T \mathbf{W} N$
 - + extensions adapted from Dwyer et al
- ◆ Prove pattern correctness once for all
- ◆ Reuse through instantiation, in matching situations





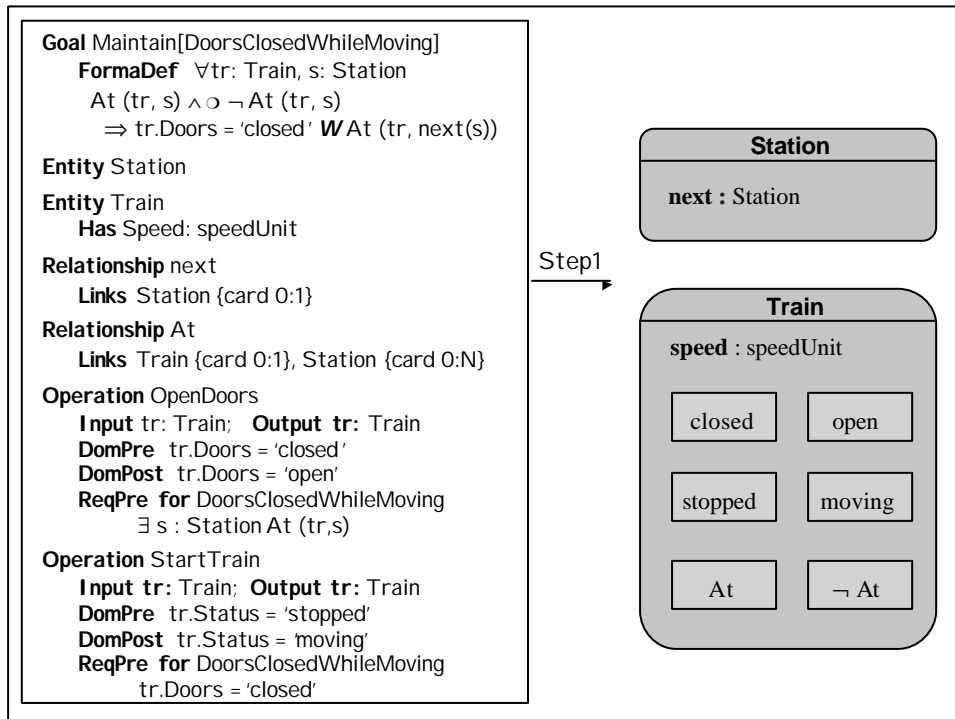


Generating state machines from goal operationalizations (2)

Step 1: Build FSM class declarations

for each $e: \text{Entity} \cup \text{Agent}$ in Object model

- create a new FSM class;
- build state attribute declaration for all behavioural attributes and relationships of e ;
- for each behavioural attribute attr identify all legal states of attr in $\text{DomPre}/\text{DomPost}$ identify additional legal states of attr in Goal

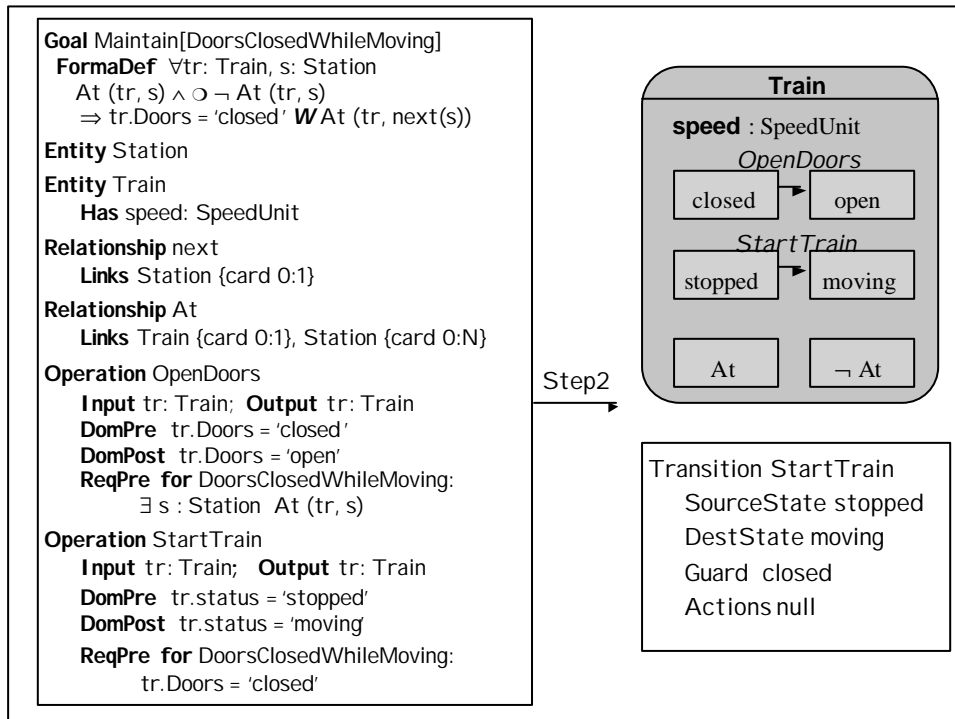


Generating state machines from goal operationalizations (3)

Step 2: Build transitions

For each op in Operation model

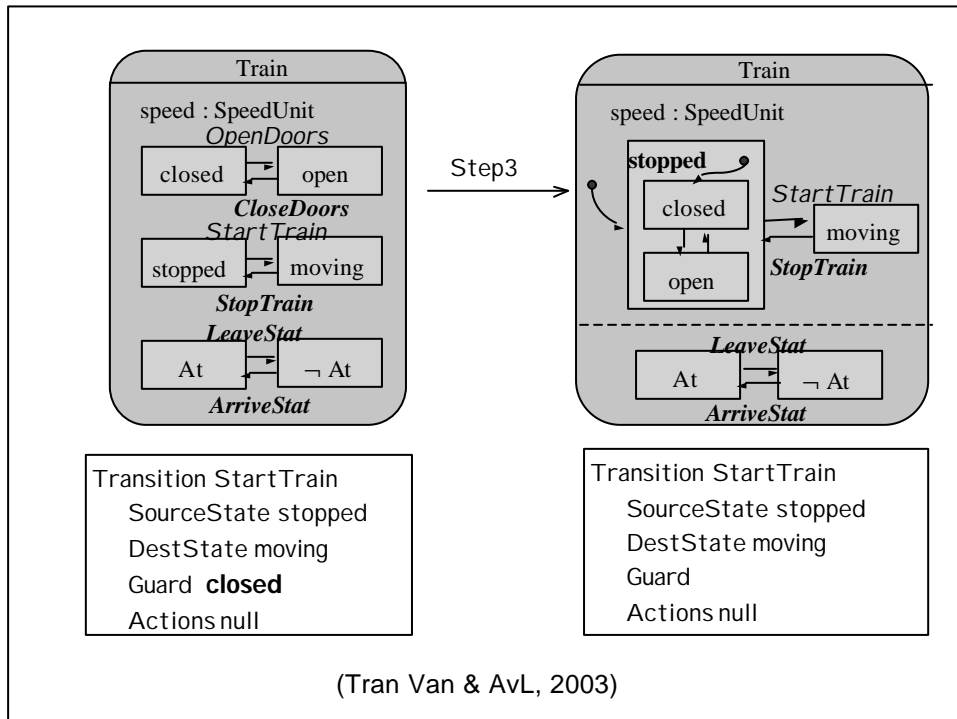
- create a new transition class;
- op.DomPre ® source state; (propositionalization)
- op.DomPost ® destination state; (propositionalization)
- op.ReqPre ® guard condition;
- op.ReqTrig ® trigger condition;
- op.DomPost , op.ReqPost ® action vector;
- host the transition;



Generating state machines from goal operationalizations (4)

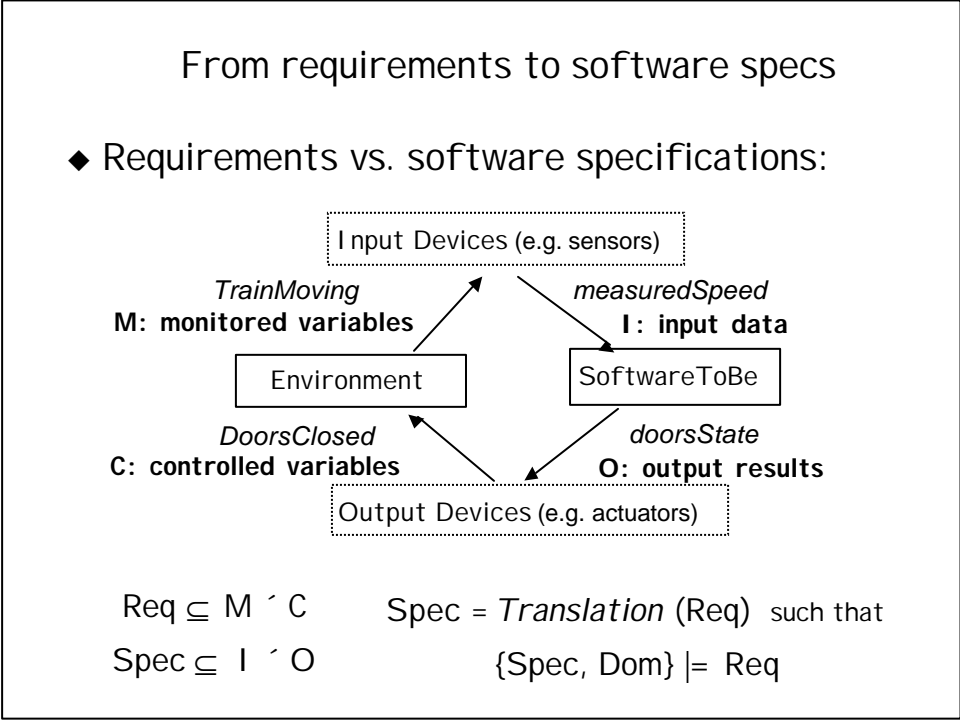
Step3: Structure the state space

- source state structuring:
if states s1, s2 have same transition to same dest state
then aggregate s1, s2 into more general state;
- guard migration:
if guard Grd on transition **T** refers to state s of hosting
object then move Grd as substate s of **T.SourceState**
(+ i/o transitions)
- additional state space structuring by analyst



Outline

- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-based reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve FRs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs



- From requirements to software specs (2)
- ◆ To map Reqs to Specs:
- translate goals assigned to software agents in vocabulary of software-to-be: input-output variables (if needed)
 - map (domain) object model elements to their images in the software's object model (if needed)
 - introduce (non-functional) accuracyGoals requiring the consistency between monitored/controlled variables in the environment & their software image (input/output variables, database elements)
 - introduce input/output agents to be responsible for such accuracy goals (sensor, actuator & other input/output devices)

From requirements to software specs (3)

◆ Example:

- Req:

MotorReversed \Leftrightarrow MovingOnRunway

- TargetSpec:

Reverse = 'enabled' \Leftrightarrow WheelPulses = 'on'

- accuracyGoals:

MovingOnRunway \Leftrightarrow WheelPulses = 'on'
expectation on wheelSensor

MotorReversed \Leftrightarrow Reverse = 'enabled'
expectation on motorActuator

Outline

- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-level reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve functional sw specs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs



Output of architecture derivation process

Structure of ...

- ◆ components, ports
- ◆ connectors
 - static: channels, roles, constraints
 - dynamic: interaction protocol
- ◆ configurations

... *to be*...

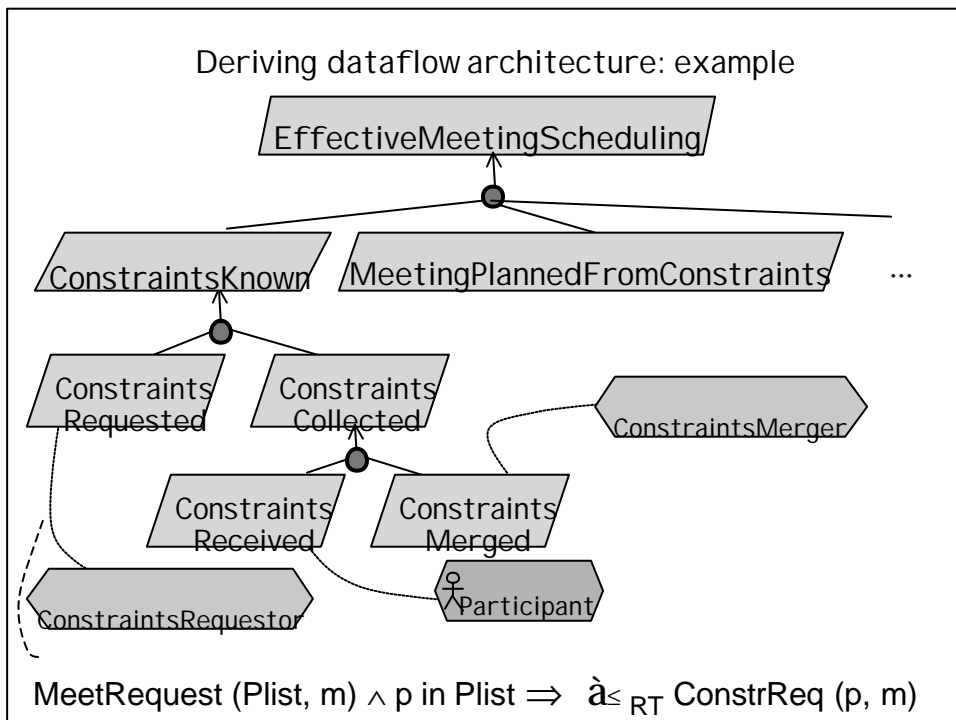
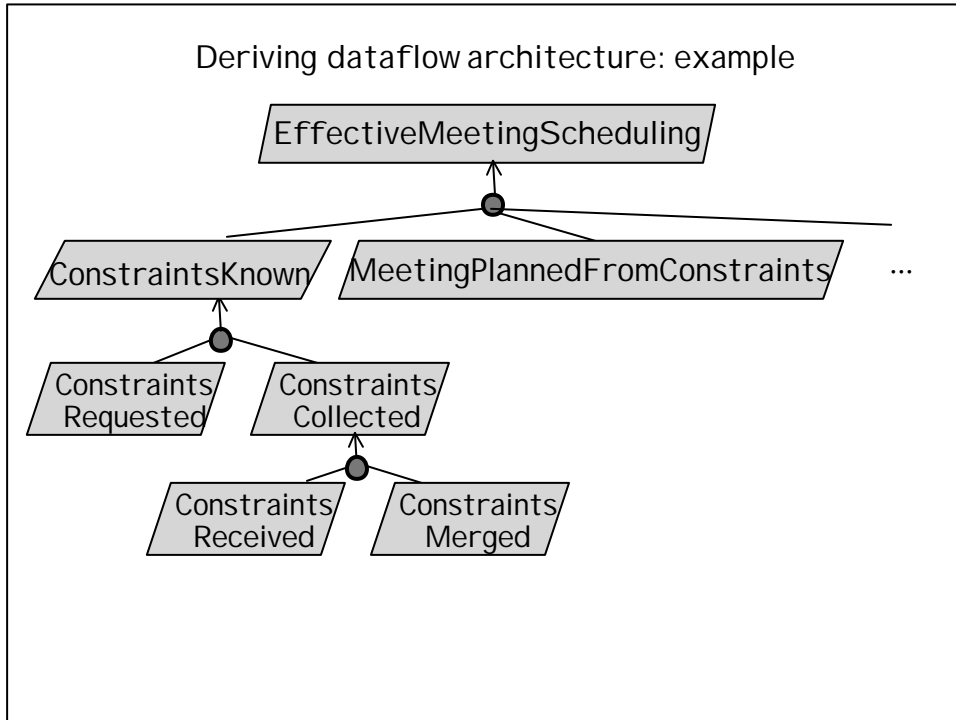
- correct: functional requirements are met
- good quality: QoS & development goals are met

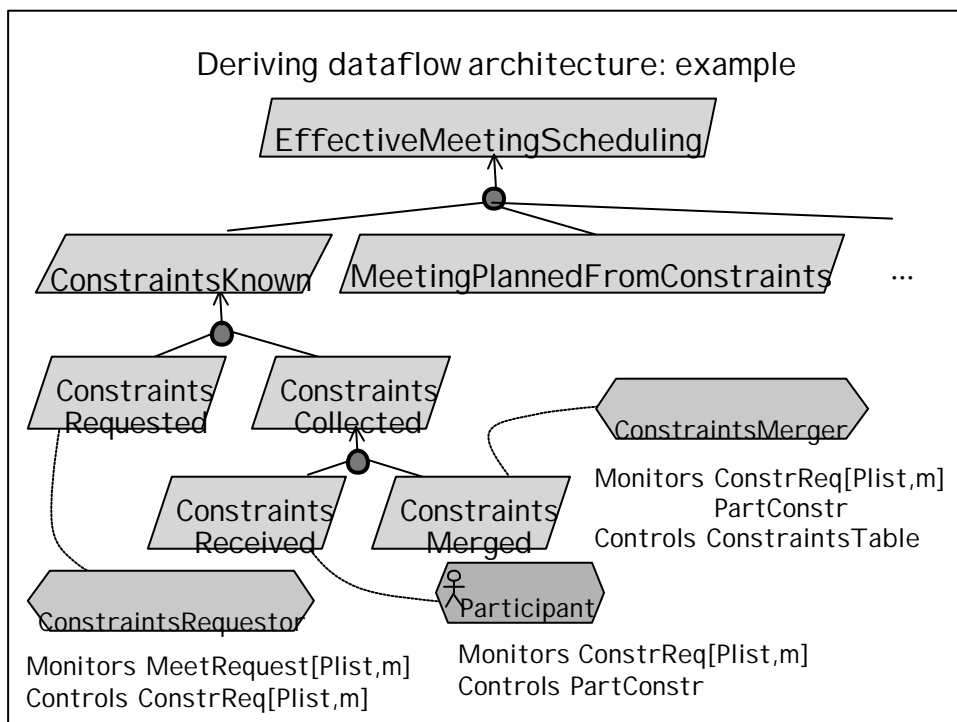
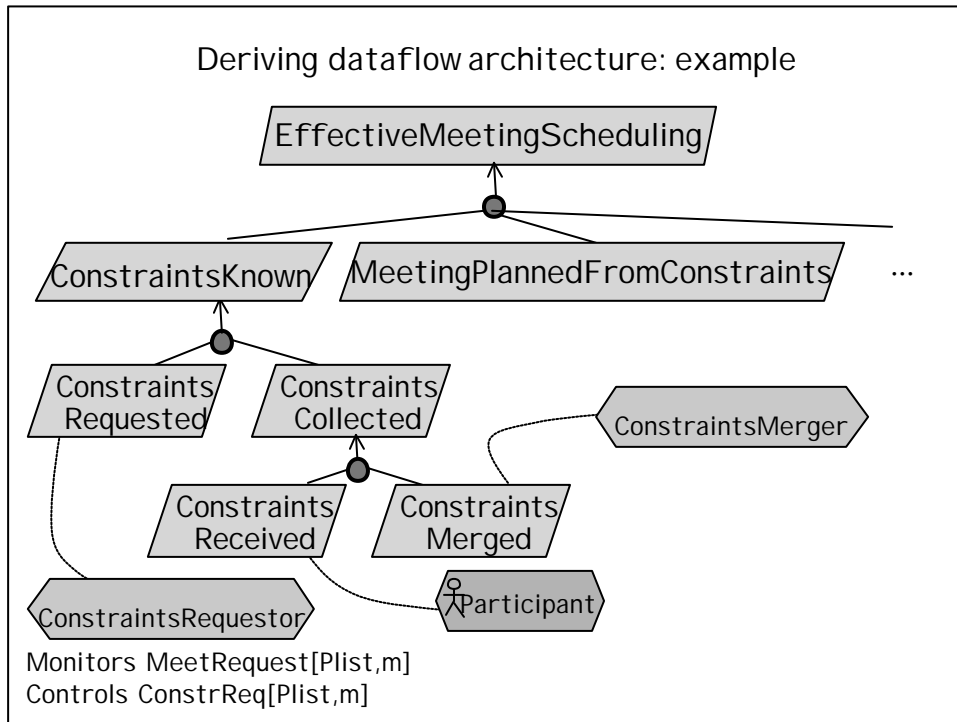
Assumption: requirements conflicts are resolved before

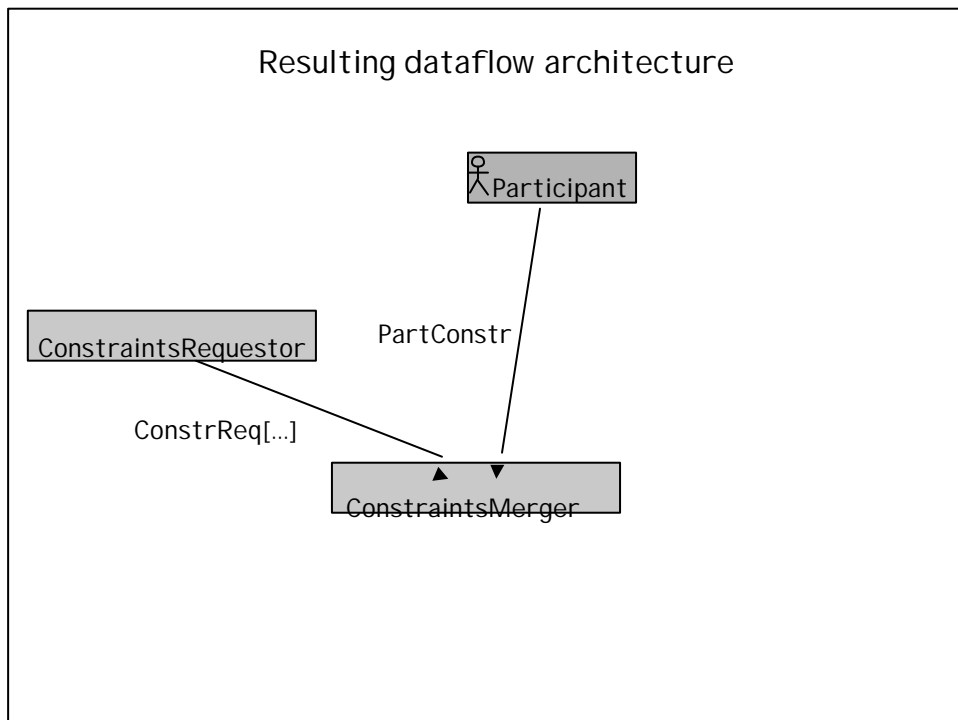
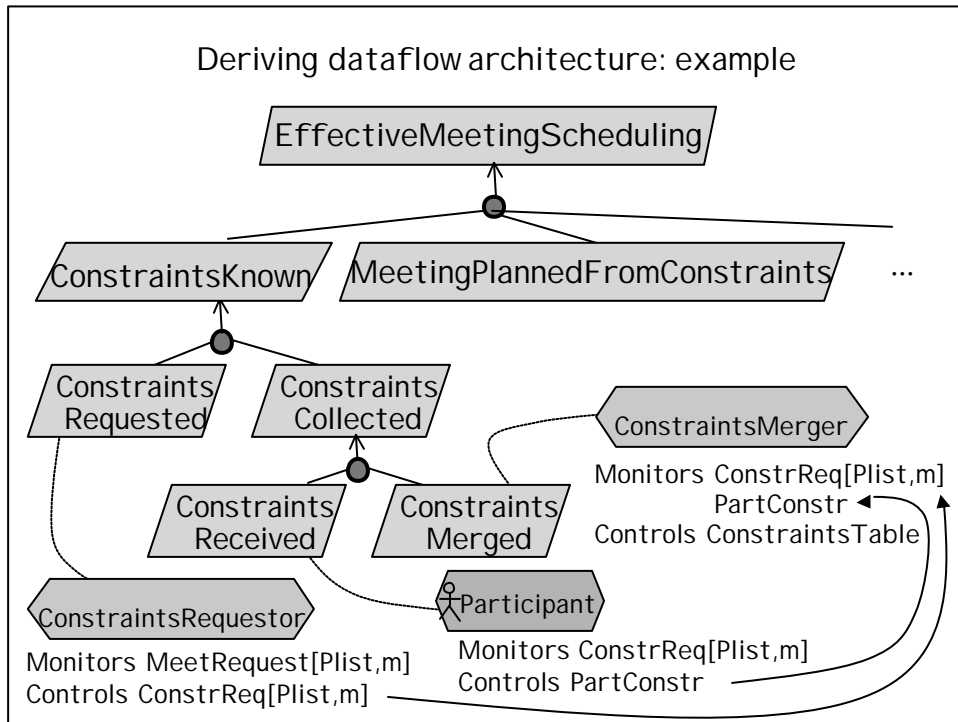
Deriving an abstract dataflow architecture

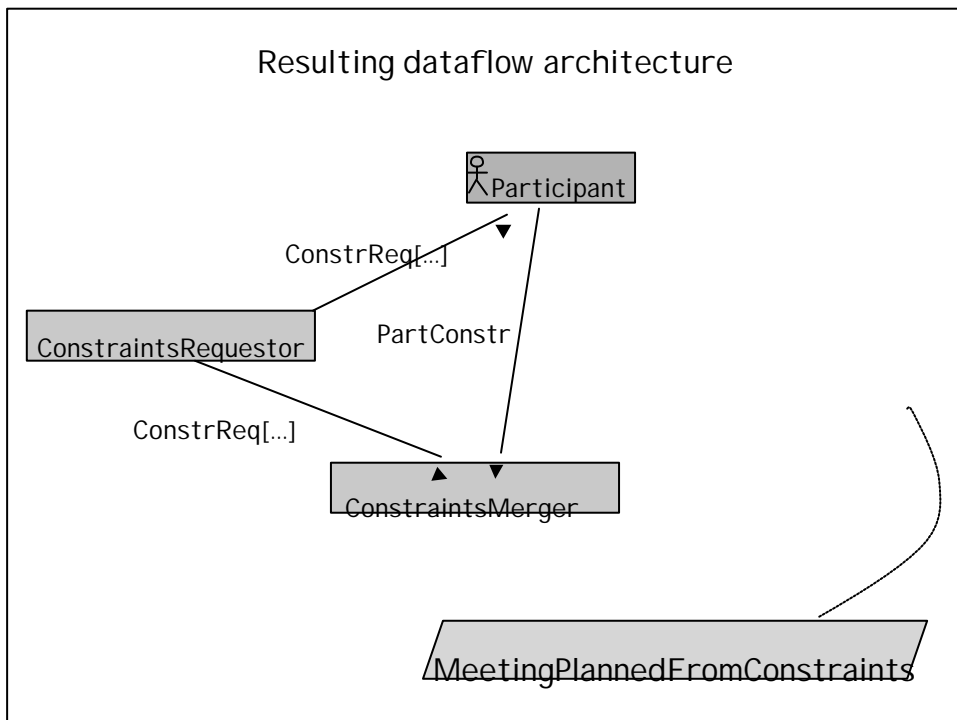
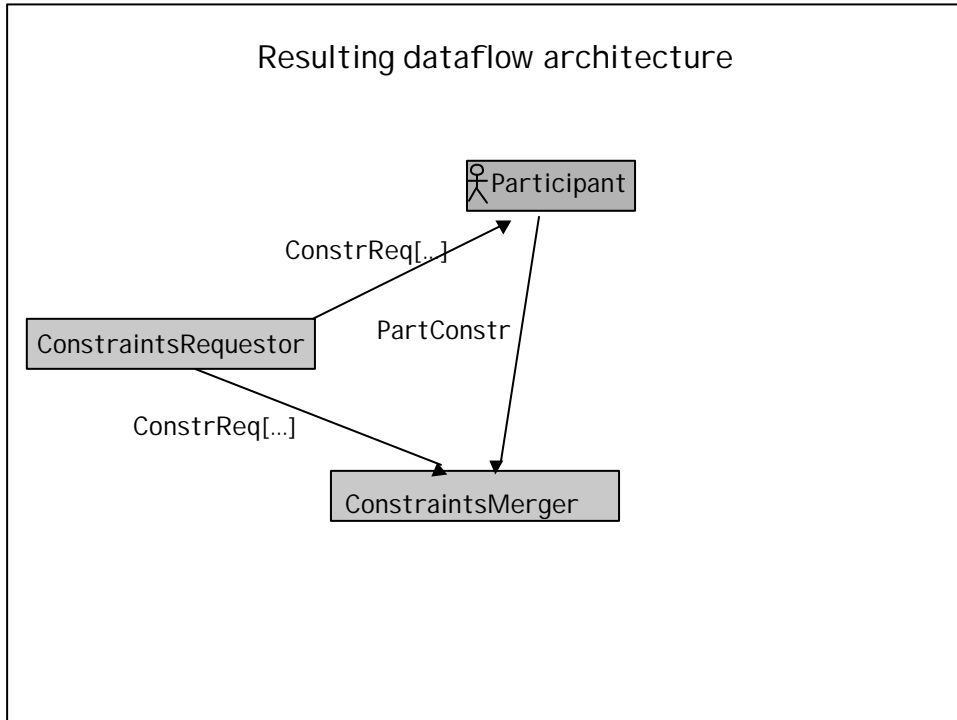
- ◆ For each "functional" or "critical" goal assigned to software-to-be:
define one dedicated component ...
 - software agent + all operations operationalizing this goal
 - interface = monitored & controlled variables in goal formulation
- ◆ Derive dataflow connector between components from data dependency links

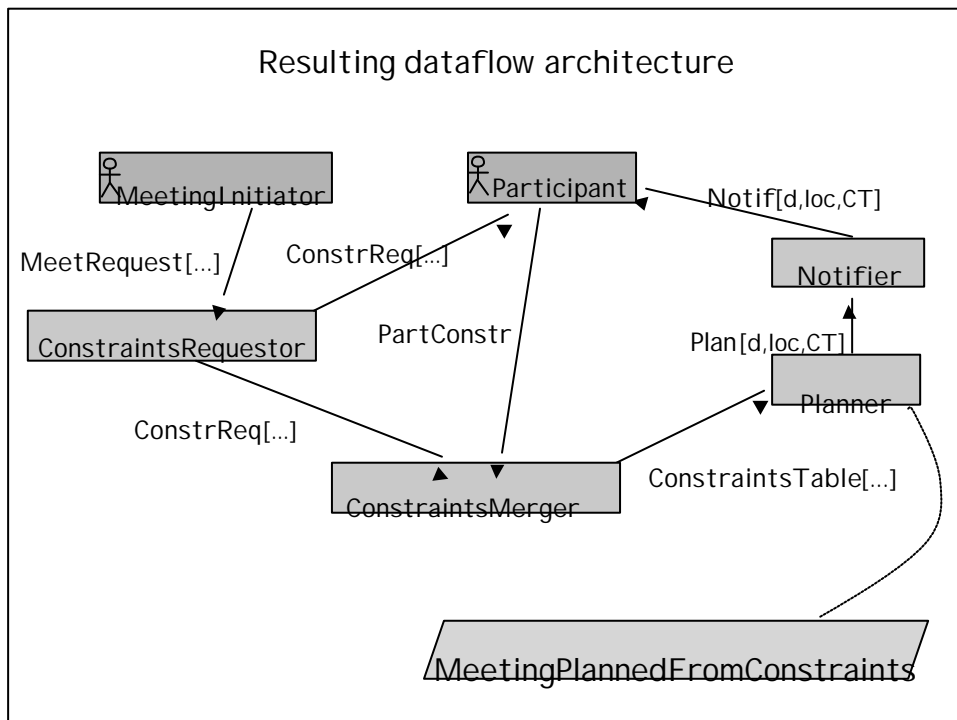
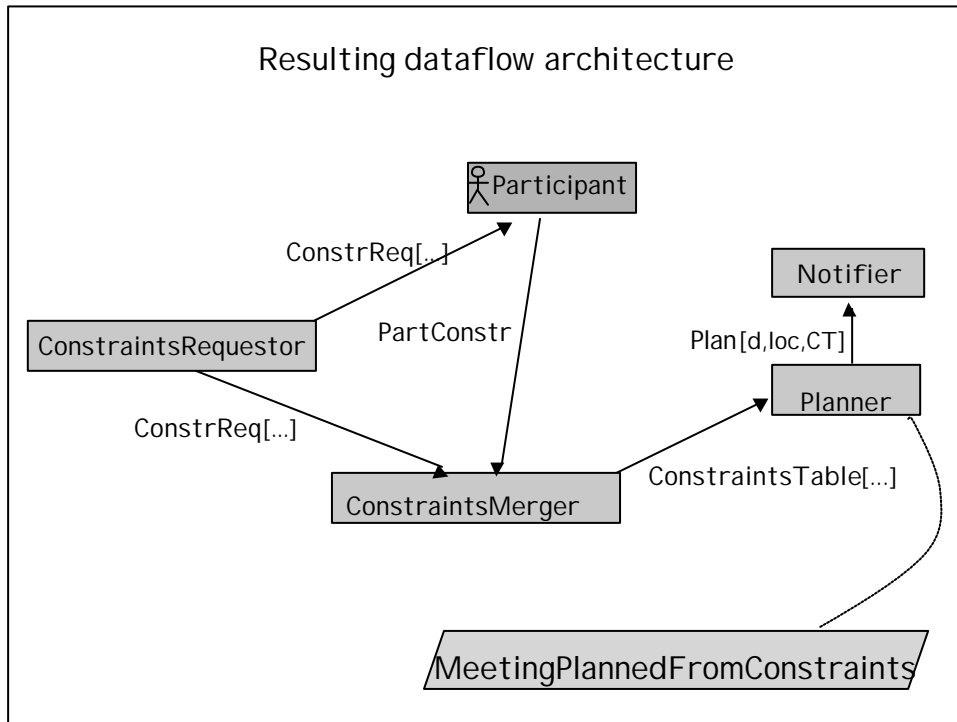
Flows $(d, C1, C2) \equiv$ Controls $(C1, d) \wedge$ Monitors $(C2, d)$











Outline

- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-level reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve functional specs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs

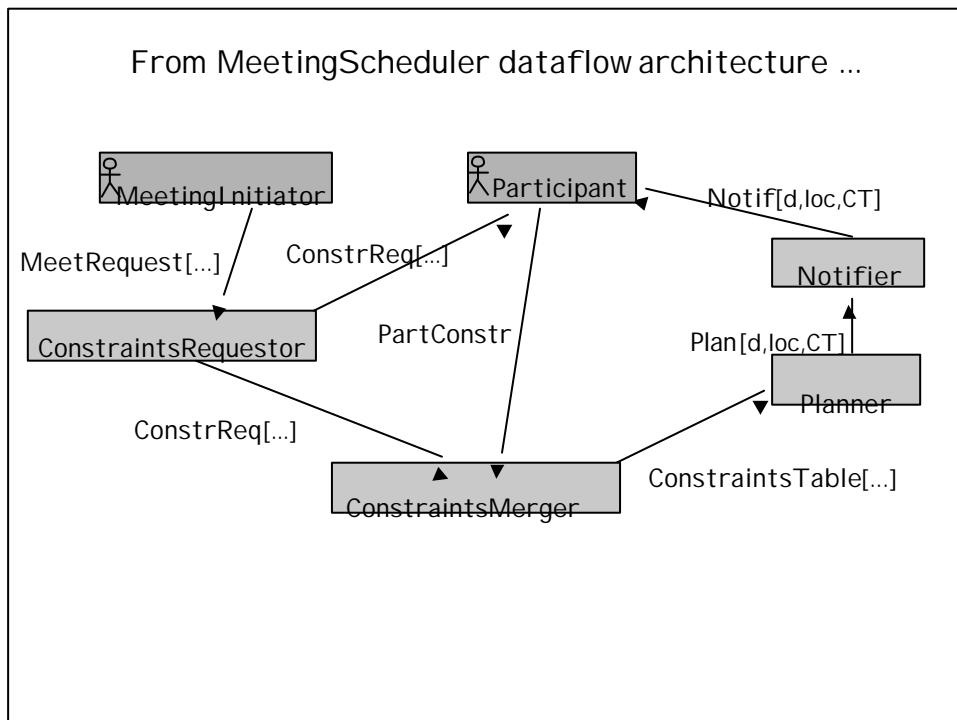
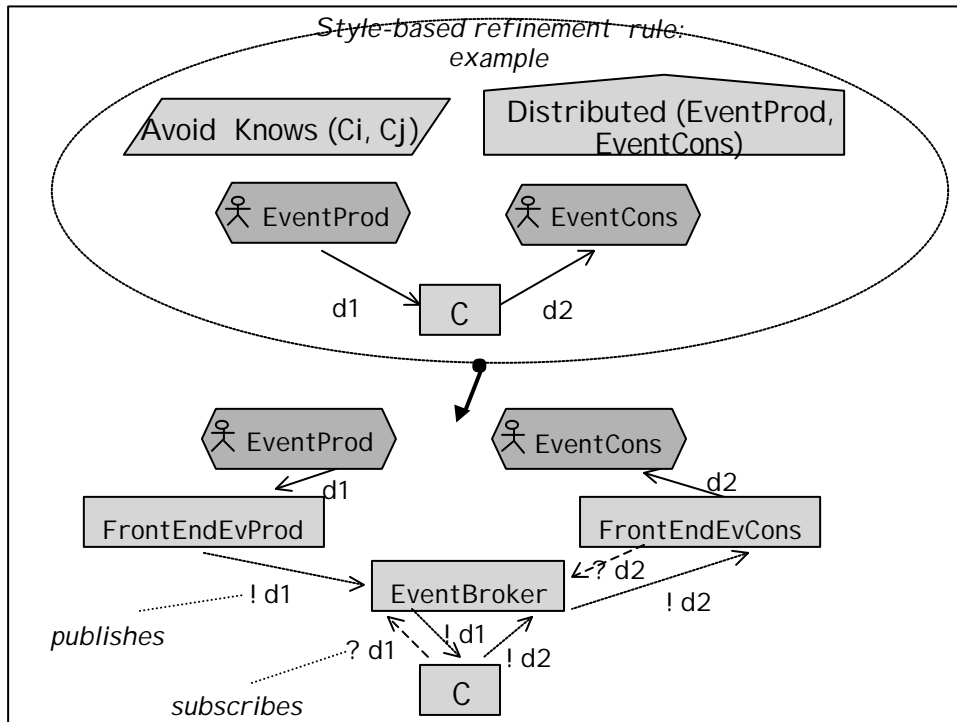
Refinement to meet architectural constraints

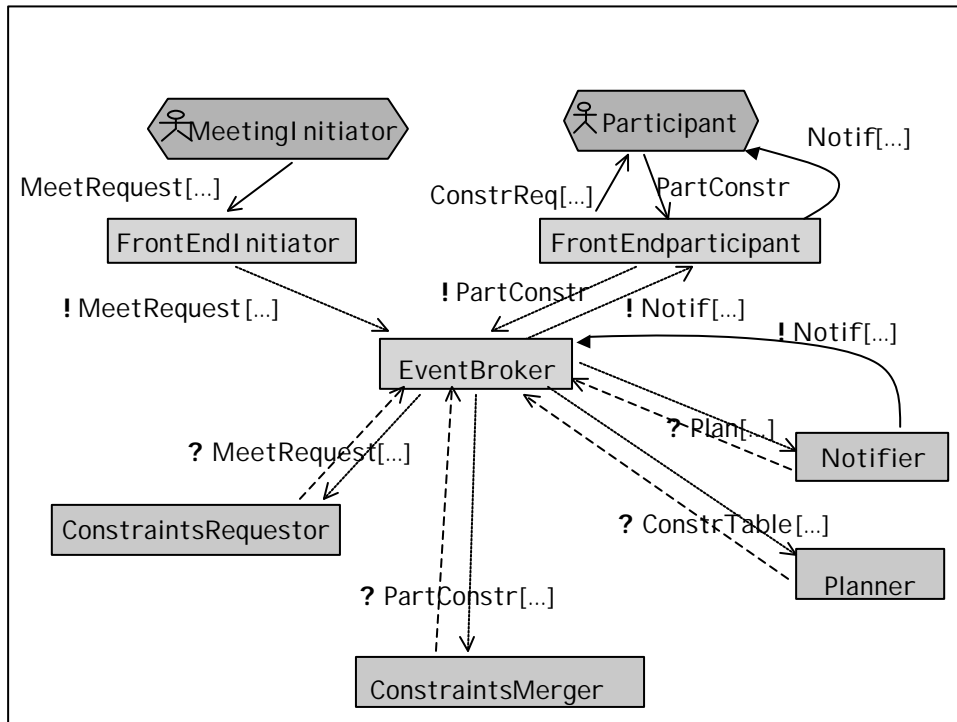
Domain-specific constraints ...

- from environment agents: features, inter-relationships
- global constraints on architectural design space
 - e.g. Meeting scheduling system:
distribution of participants, meeting initiator

Idea:

- ◆ Document styles by rules
 - (domain conditions, target_NFR) → effect
- ◆ Apply rule *matching architectural constraint*
- ◆ Proof obligation: rule application must preserve properties of components & connectors (e.g., dataflows)





Outline

- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
 - Building goal-oriented requirements models
 - Intertwining between late RE & early AD
 - Goal-level reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software architecture
 - Derivation of abstract dataflow architecture to achieve functional specs
 - Style-based refinement to meet architectural constraints
 - Pattern-based refinement to achieve NFRs

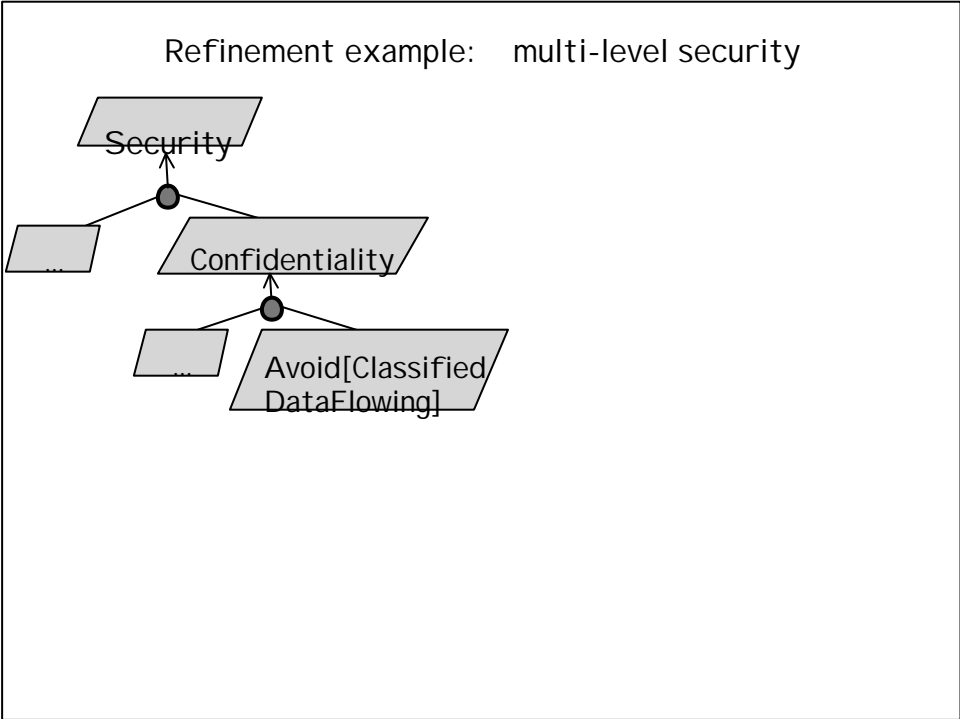
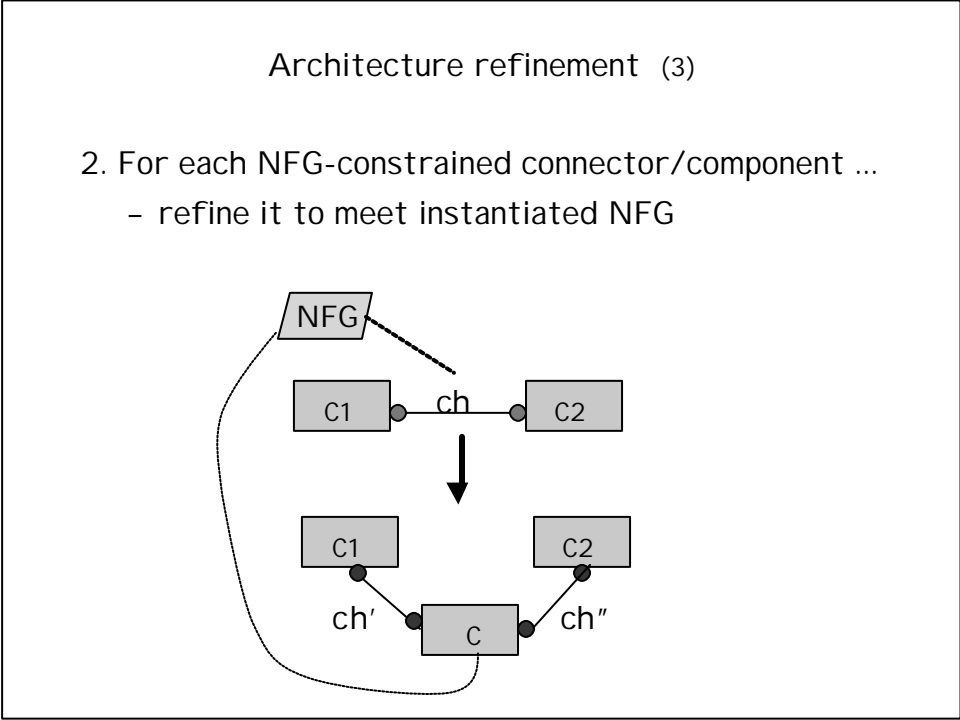
Architecture refinement

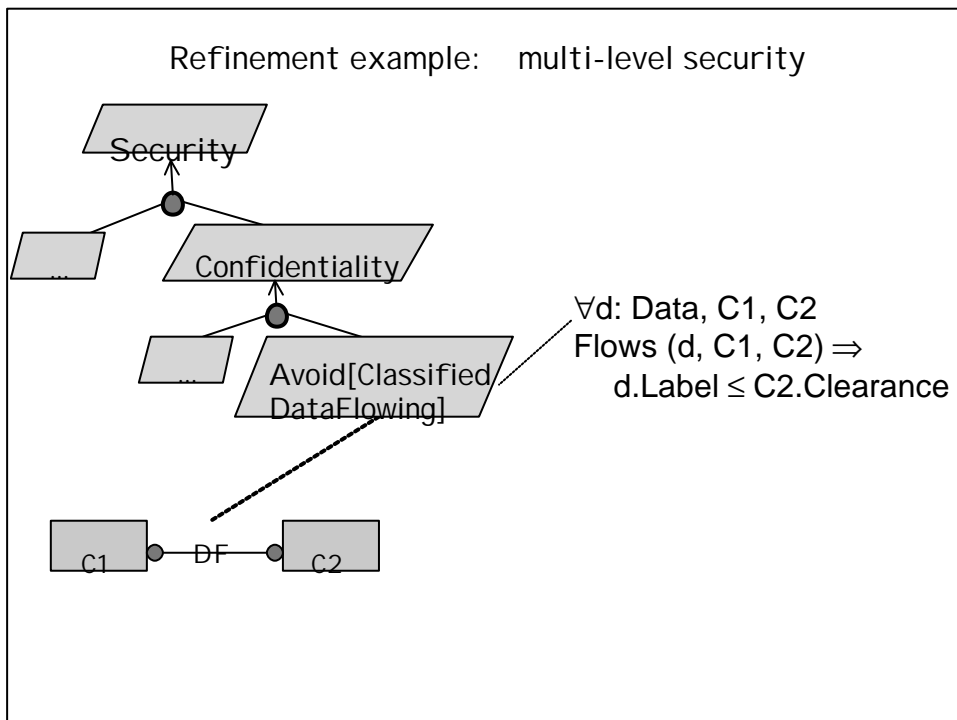
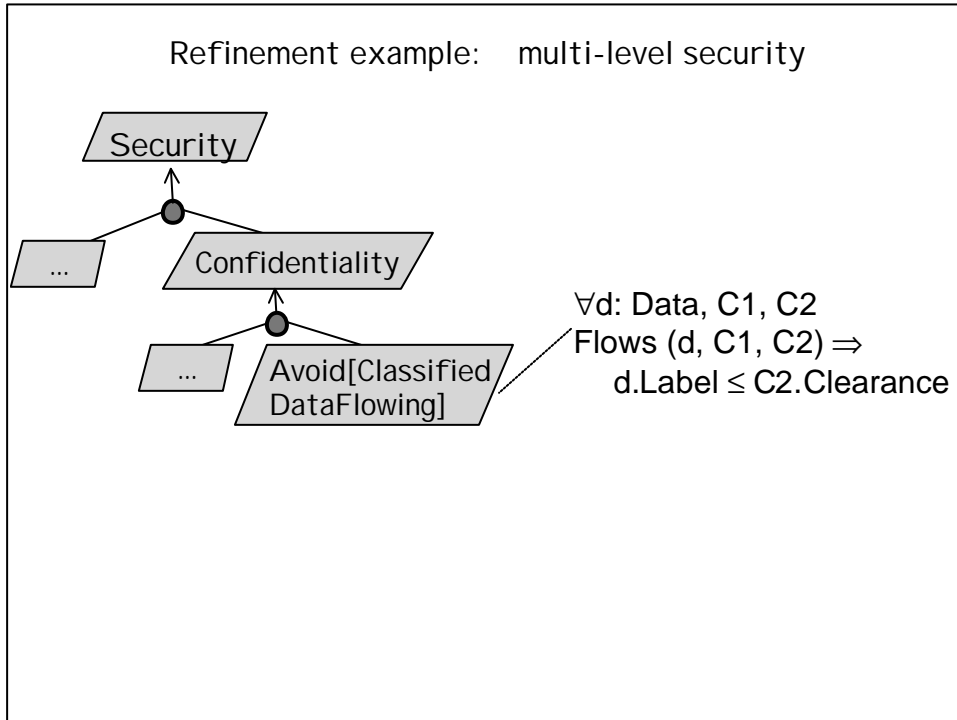
- ◆ Many non-functional goals impose constraints on component interaction
 - Accuracy (C1,C2): data consistency
 - Confidentiality (C1,C2): limitation on info flow
 - Usability (C1,C2): requirement on presentation, dialog
 - etc: MinCoupling (C1,C2), InfoHidden (C1, C2), Interoperable (C1,C2), ...
- ◆ Some NFGs impose constraints on single component
 - MaxCohesion (C): fine-grained functionality

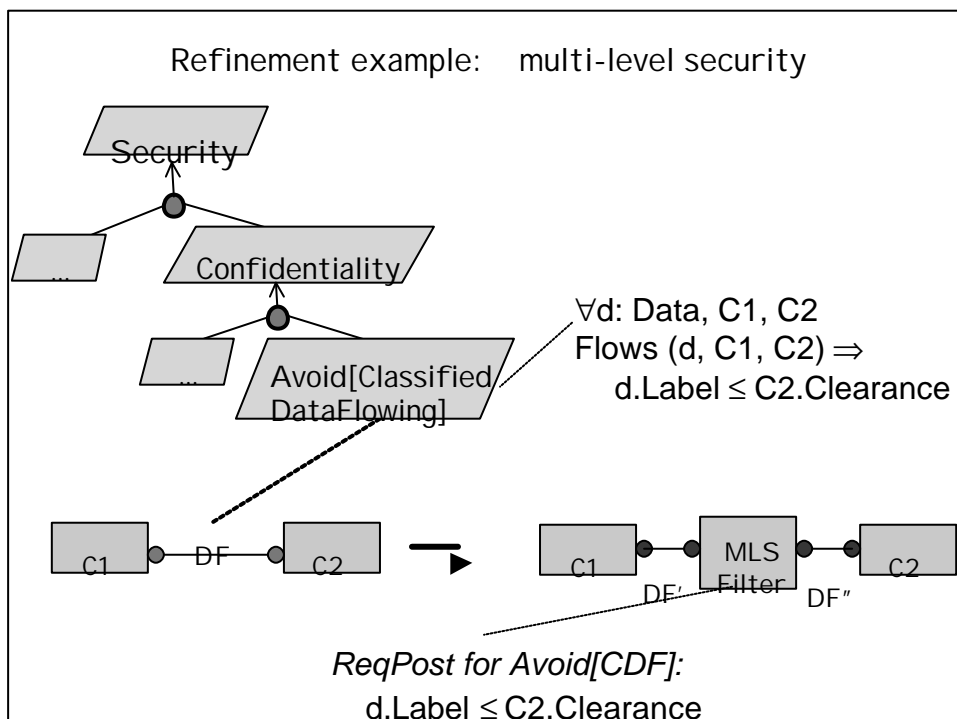
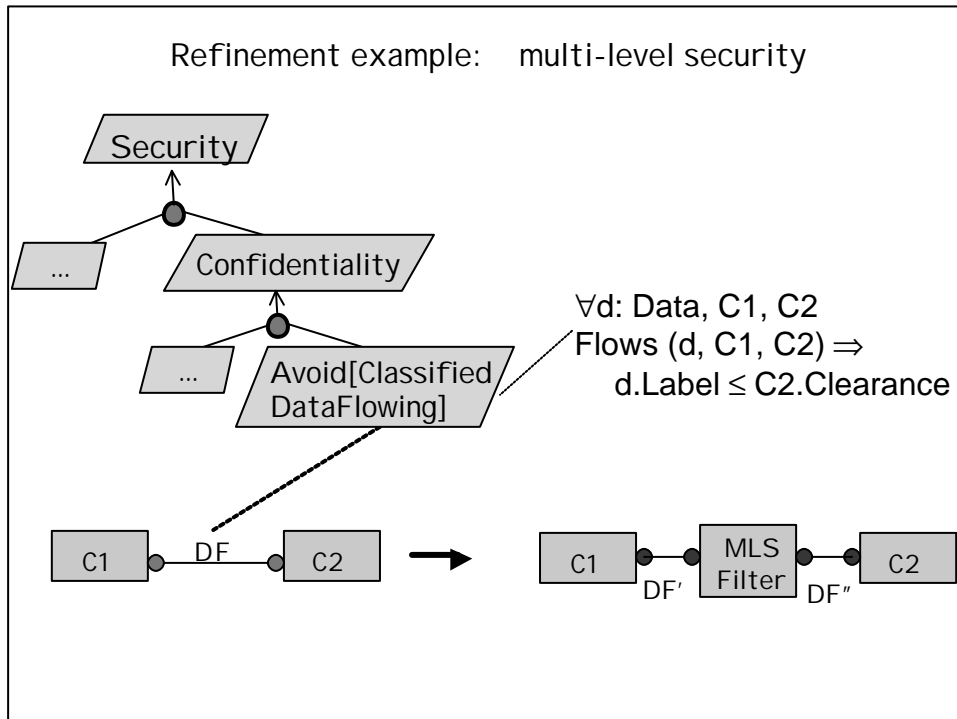
Architecture refinement (2)

1. For each terminal NFG in goal refinement graph ...

- identify all connectors/components constrained by it
- instantiate it to those connectors/components





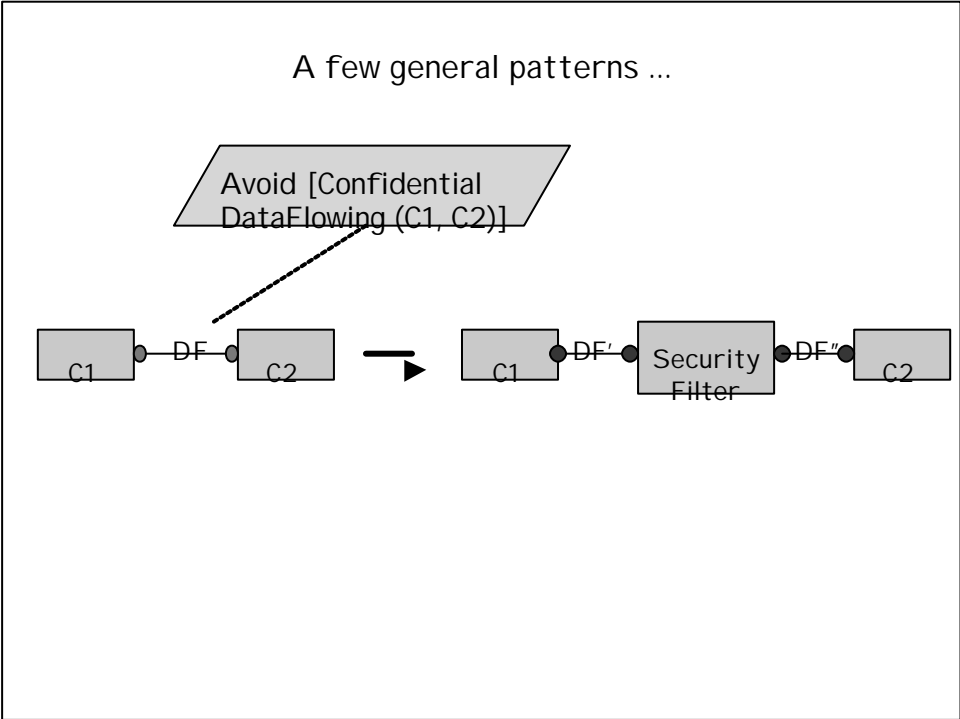


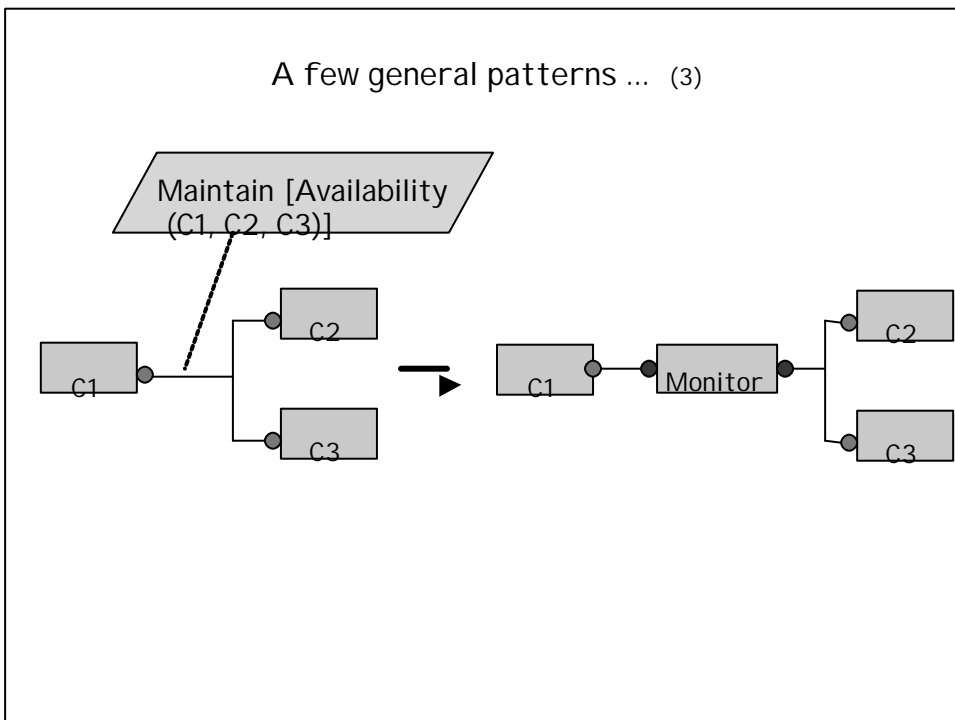
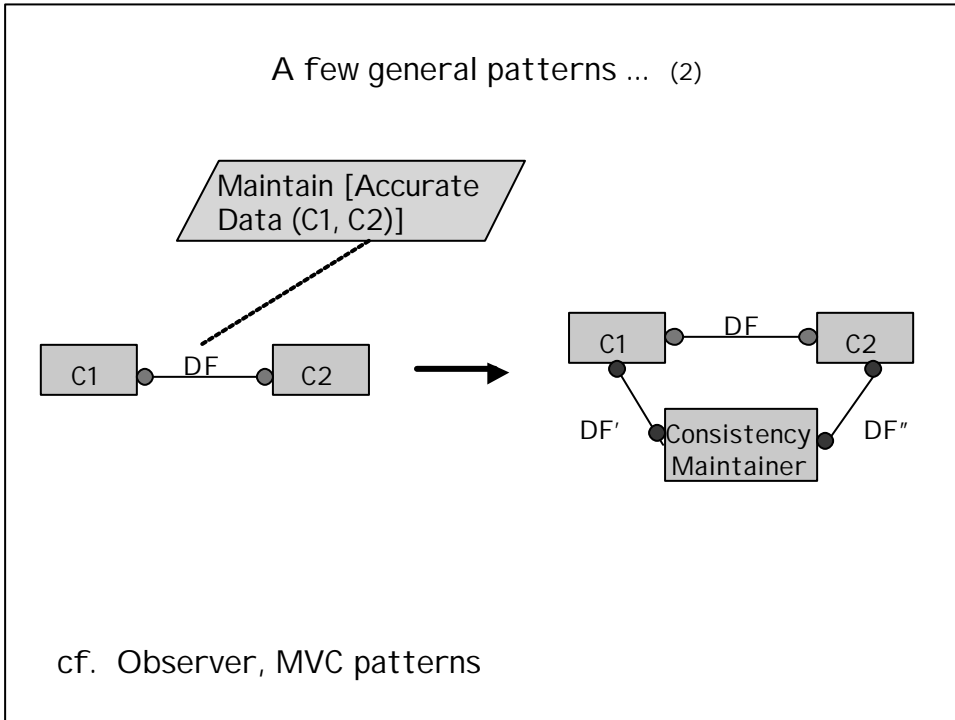
Architecture refinement (4)

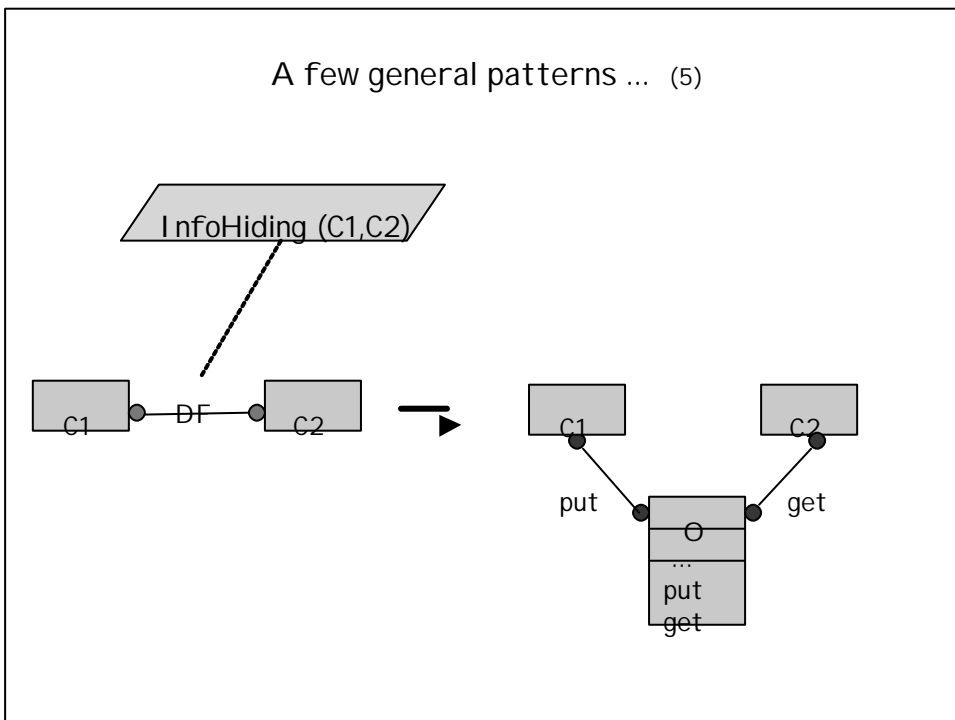
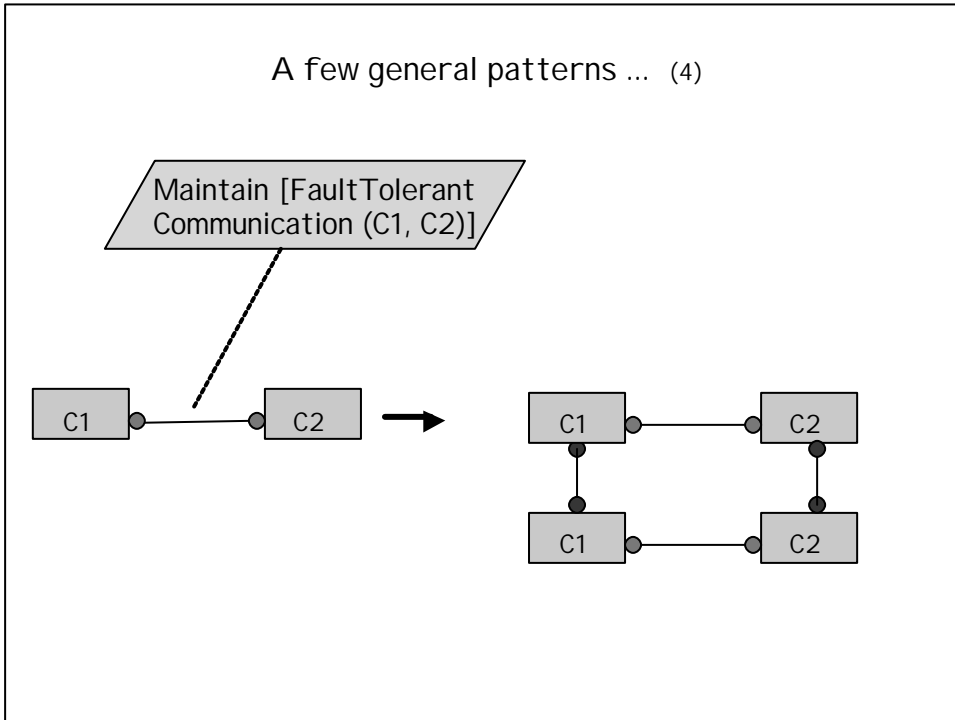
2. For each NFG-constrained connector/component ...
- refine it to meet instantiated NFG ...

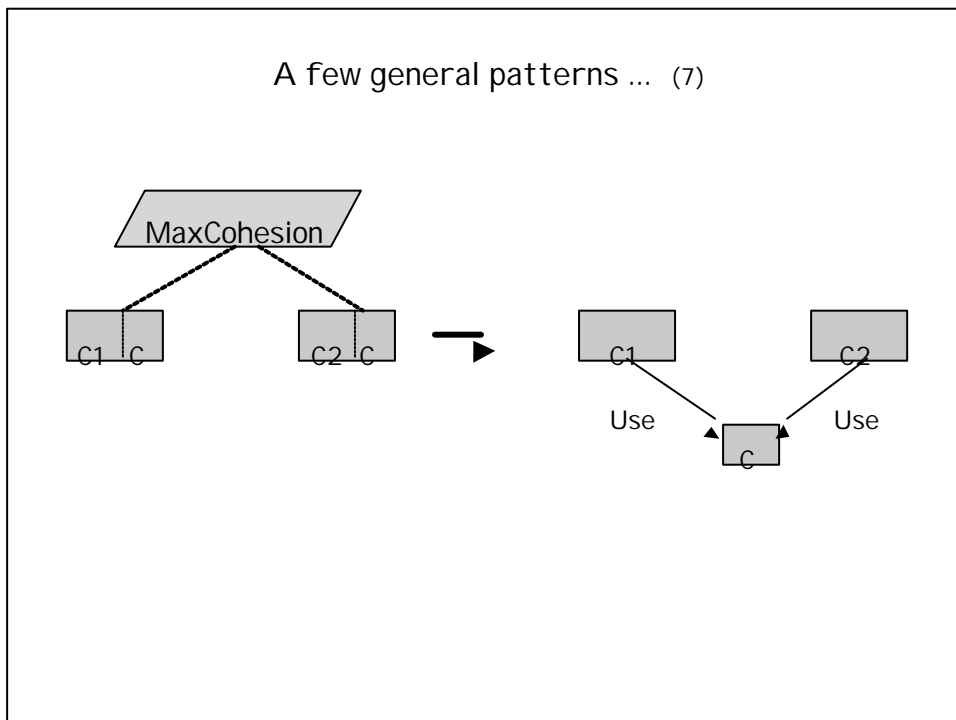
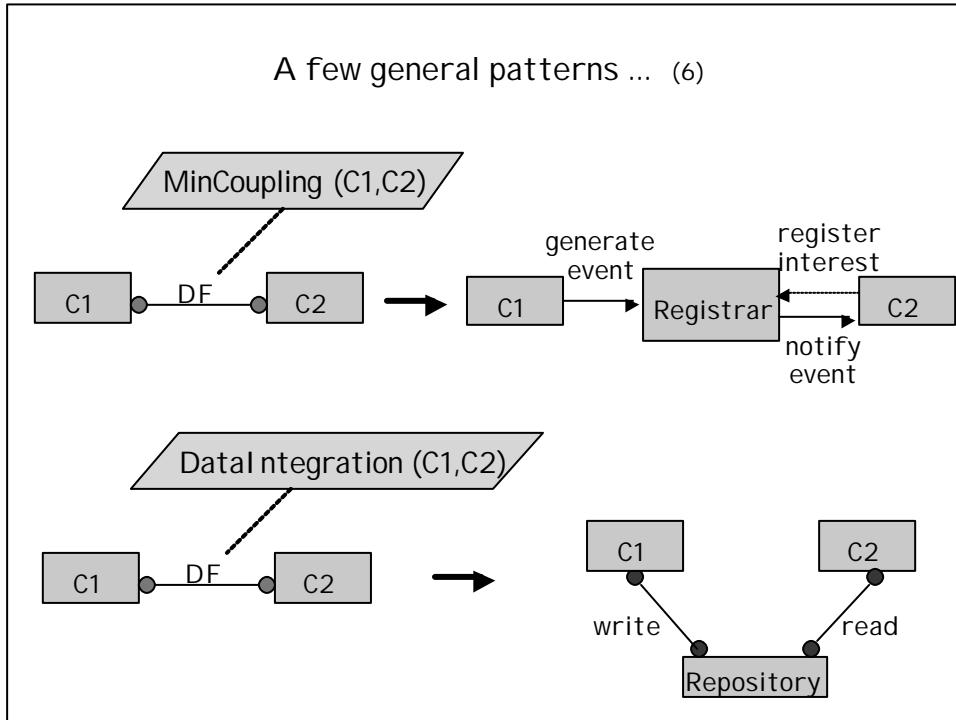
by use of architectural refinement patterns:

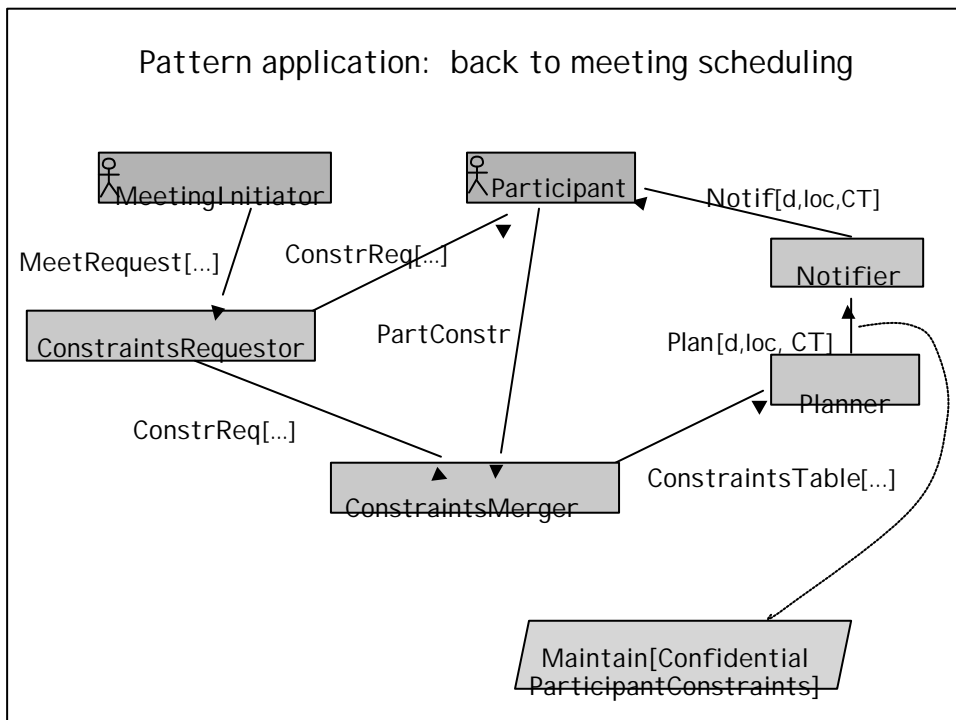
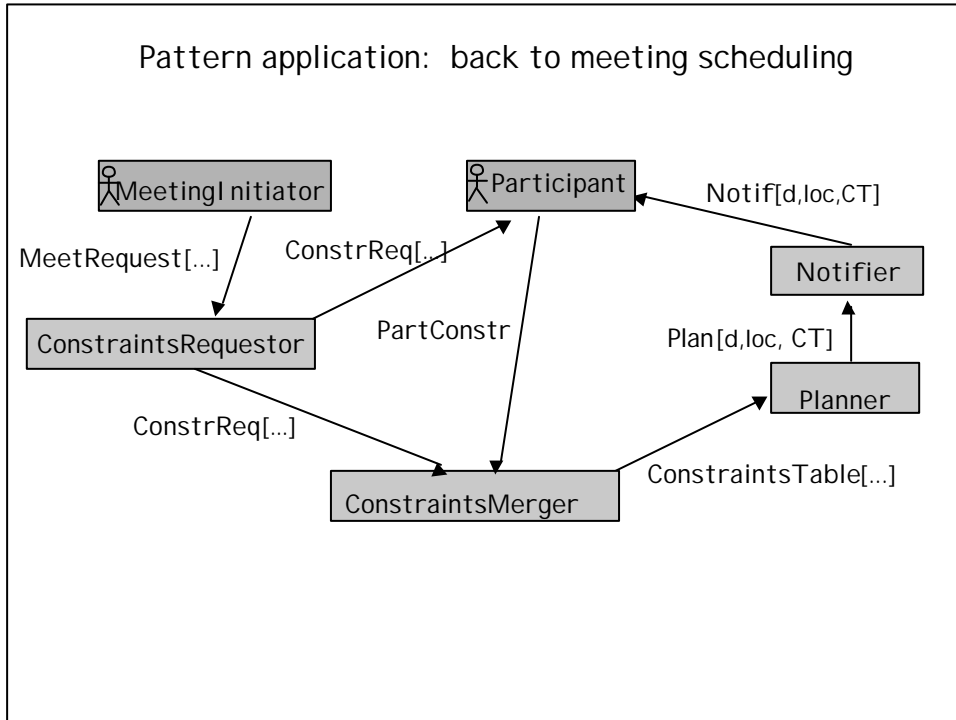
- catalog of refinement patterns
- each pattern is annotated by underlying design goals & tradeoff documentation (cf. [Gross&Yu'01])
- pattern selection by goal matching (conflict resolution by goal prioritization based on tradeoff analysis à la NFR)

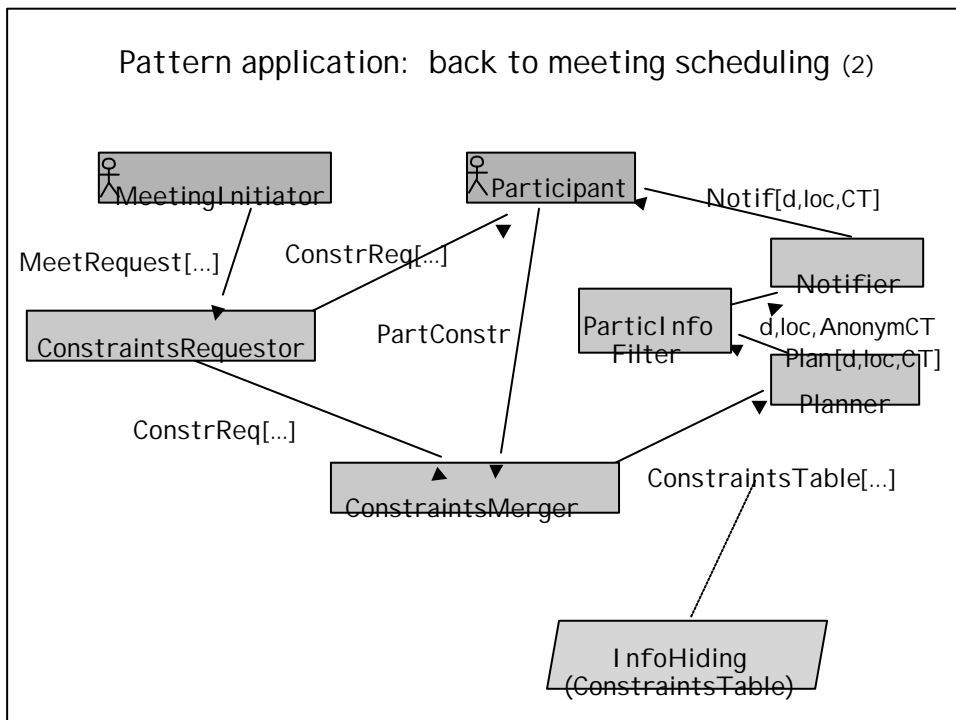
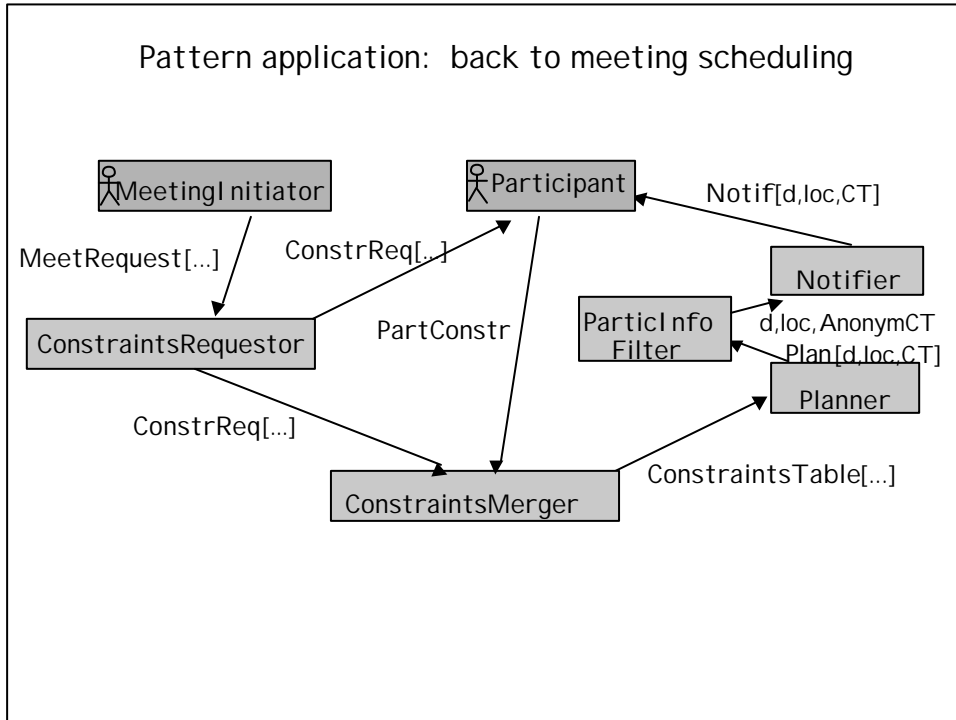


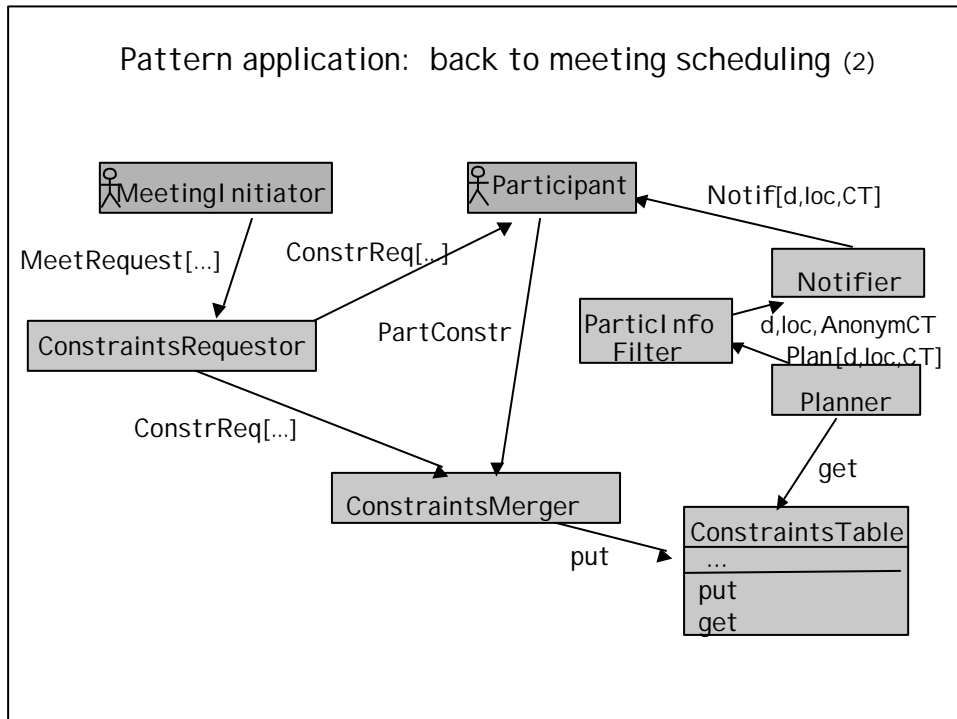












Conclusion

- ◆ Much room for incremental analysis of partial models at *goal level*
- ◆ Derivation of architecture from requirements ...
 - systematic
 - incremental
 - locality principle; compositional
- ◆ Refined connectors/components explicitly linked to non-functional goals

β

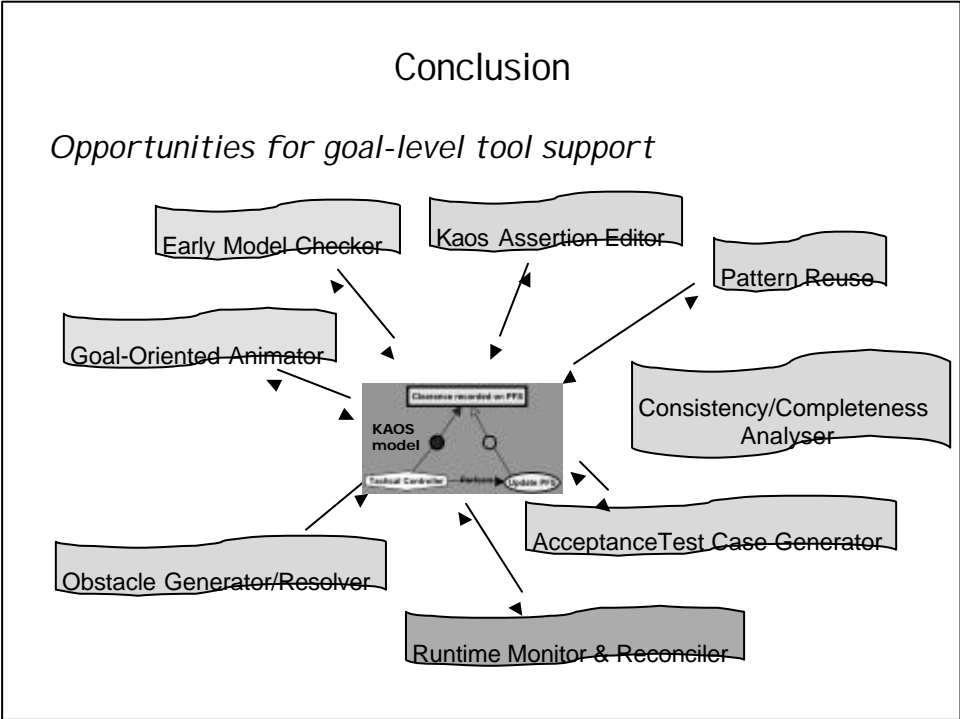
view extraction through architectural net queries:
security view, accuracy view, reusability view, ...

Conclusion

- ◆ Much room for incremental analysis of partial models at *goal level*
- ◆ Derivation of architecture from requirements ...
 - systematic
 - incremental
 - locality principle; compositional
- ◆ Refined connectors/components explicitly linked to non-functional goals

β

view extraction through architectural net queries:
security view, accuracy view, reusability view, ...



Limitations & further work

- ◆ Only refinement-based:
 - no bottom-up propagation of middleware requirements
 - β
 - need for complementary abstraction patterns
- ◆ No derivation of interaction protocols
 - β
 - integration of previous work on synthesis of concurrent interaction schemes from goal-based invariants
- ◆ RE_NET: towards requirements/architecture co-design & co-evolution...
 - at development time
 - at run time

For more info ...

- ◆ Papers:
 - GOOGLE Axel van Lamsweerde goals KAOS
- ◆ Forthcoming book