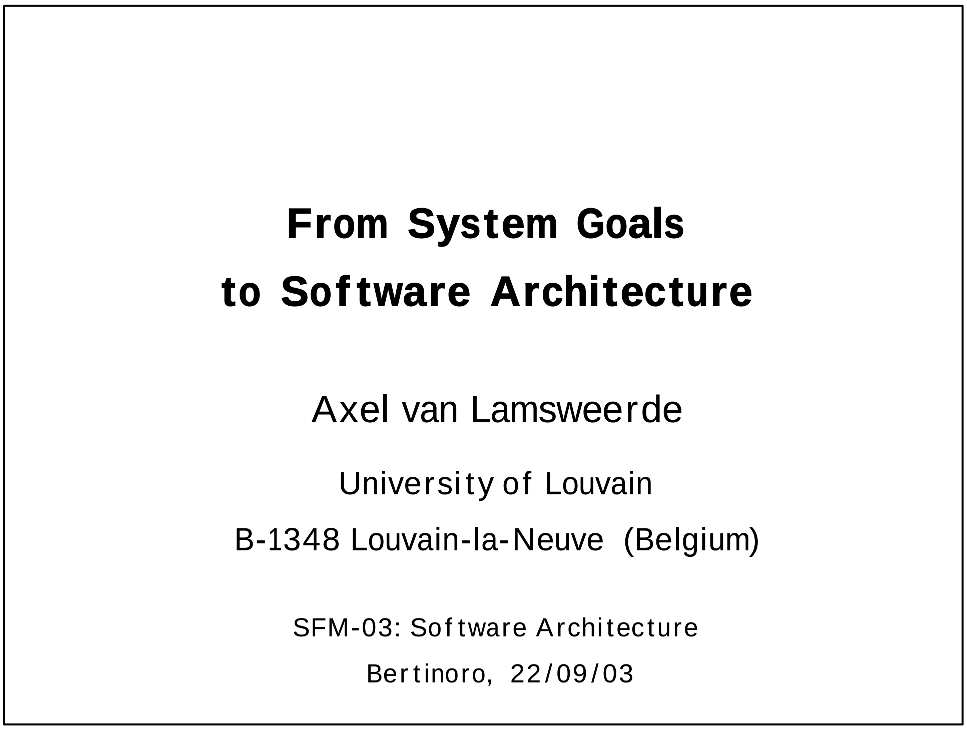


From System Goals to Software Architecture

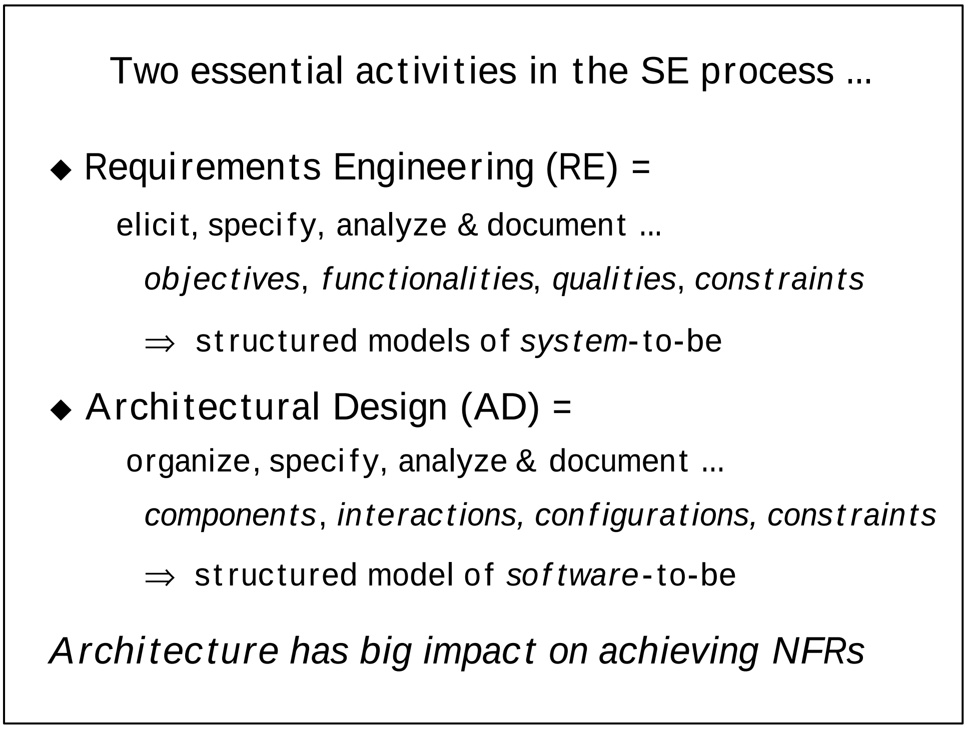
Axel van Lamsweerde

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

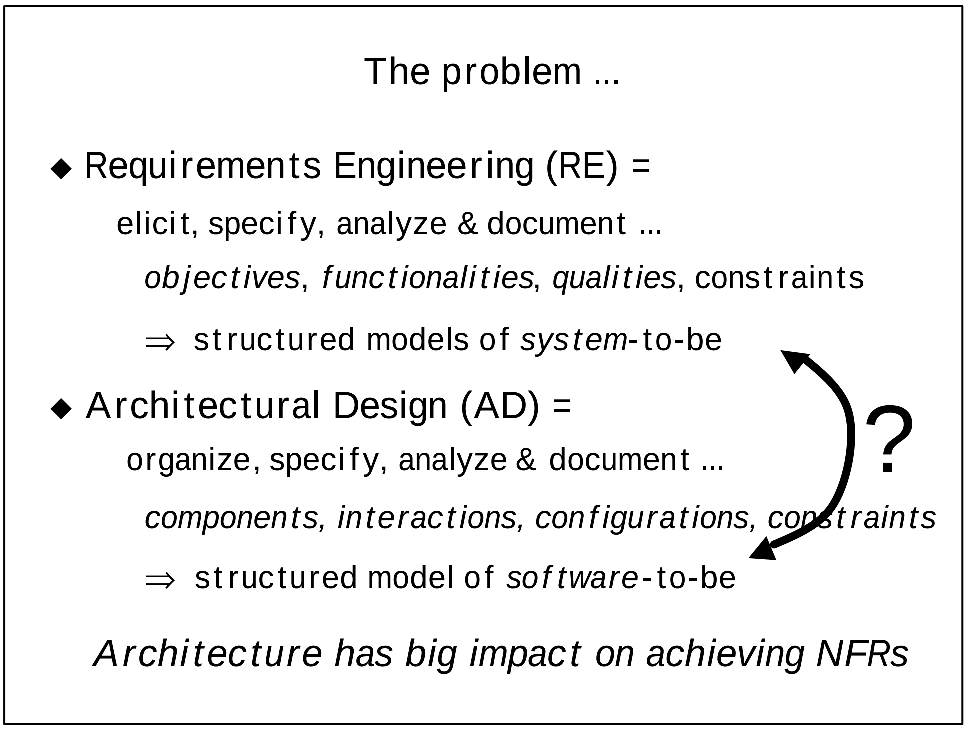
- Architectural Design (AD) =

  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

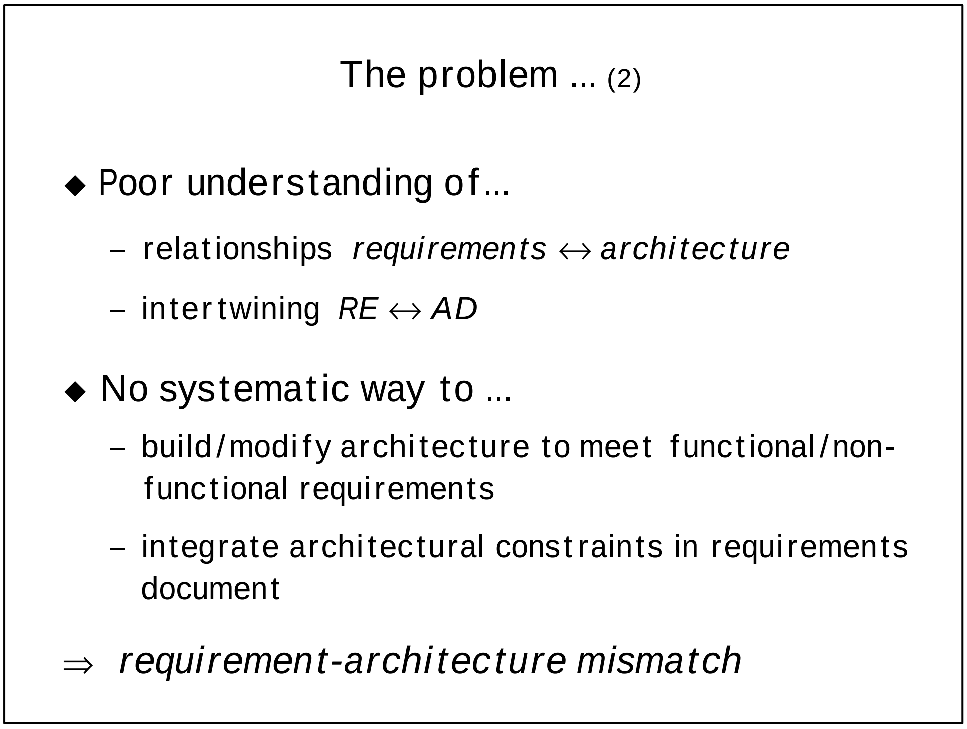
- Architectural Design (AD) =

  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

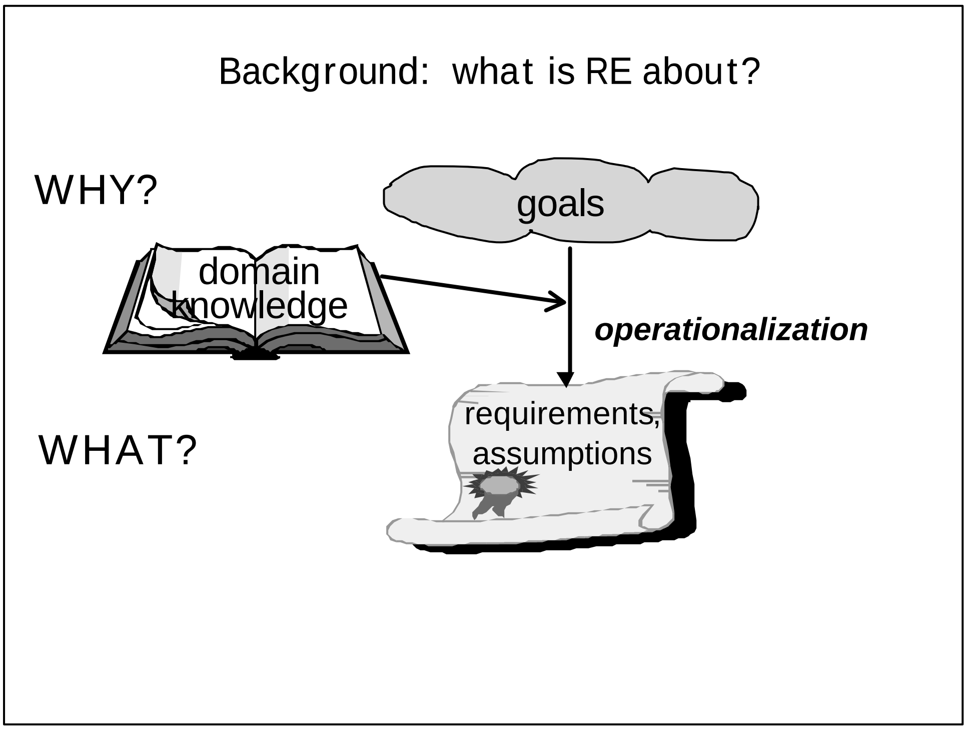
- Support requirements/architecture co-design/co-evolution
- Support architecture derivation from requirements models & software specs
- Make derivation process...
  - systematic, incremental
  - leading to provably/arguably correct & "good" architecture
  - highlighting architectural views (e.g. security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

ß

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-levl 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?

WHY?

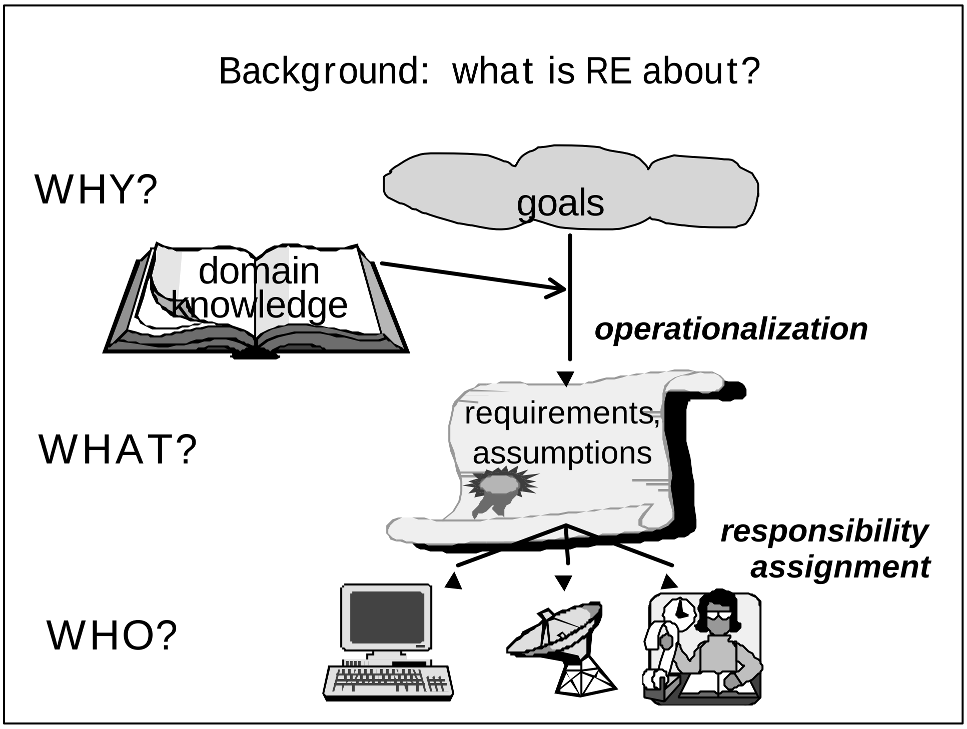
goals

domain knowledge

operationalization

WHAT?

requirements, assumptions



Background: what is RE about?

WHY?

goals

domain knowledge

operationalization

WHAT?

requirements, assumptions

responsibility assignment

WHO?

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)

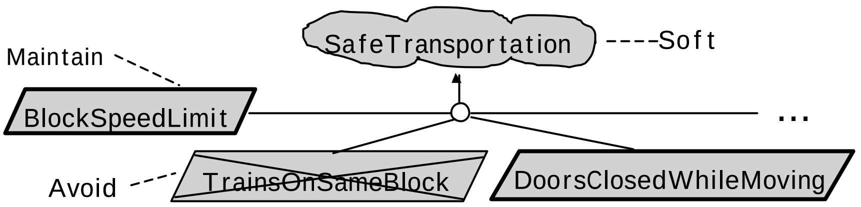
- Different types of goals ...
  - SoftGoal achievement cannot be established in clear-cut sense

    ® 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 $G_1, ..., G_n$ iff achieving $G_1, ..., G_n$ contributes to achieving *G*

  the set $\{G_1, ..., G_n\}$ is called refinement of *G*
  G_i is said to contribute positively to *G*

- The set $\{G_1, ..., G_n\}$ is a complete AND-refinement of *G* iff $G_1, ..., G_n$ are sufficient for achieving *G* in view of known domain properties

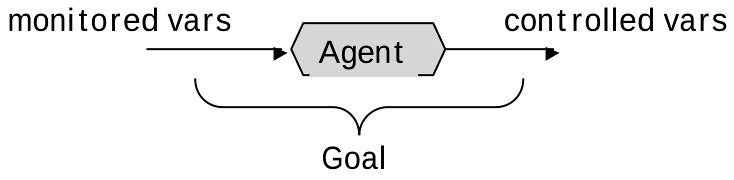
  $\{G_1, ..., G_n, \text{Dom}\} \models G$

- Goal *G* is OR-refined into refinements $R_1, ..., G_m$ 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

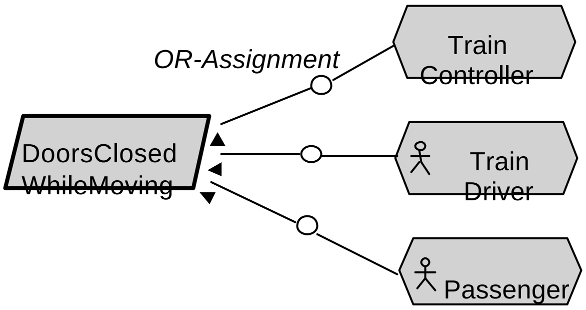


Goals need to be refined until assignable to single agents

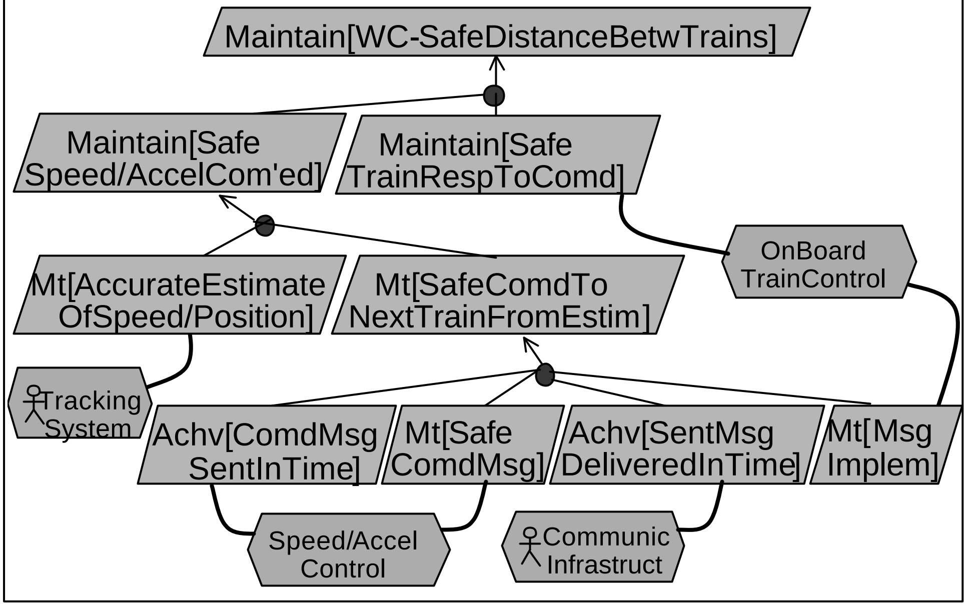
Background: goal-oriented RE (7)

- Agent responsibility:

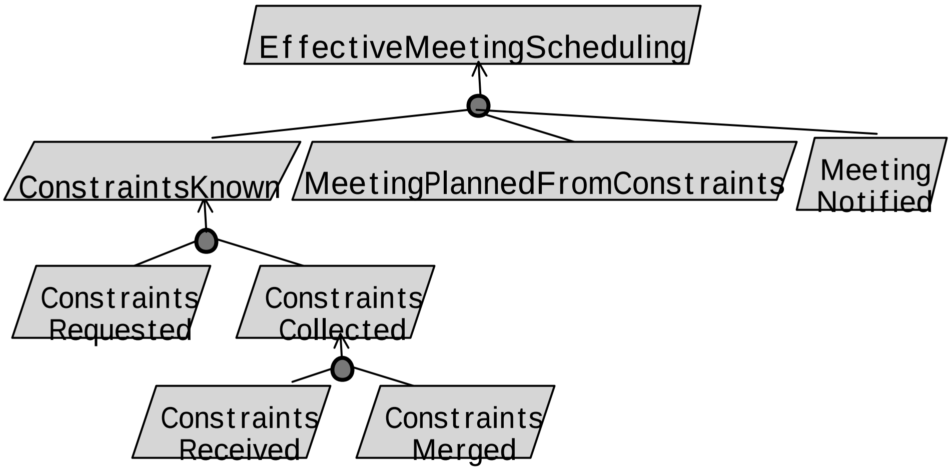
  G is assignable to Ag iff G is *realizable* by Ag



Modeling goals & responsibilities

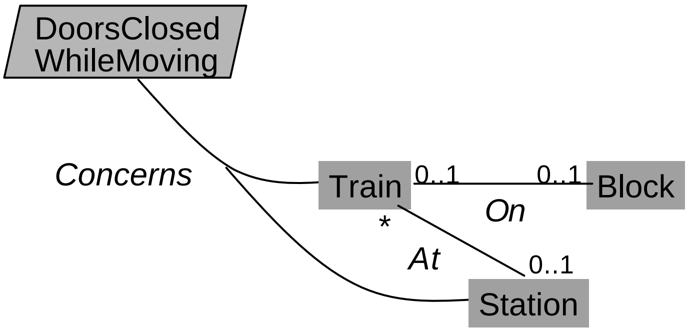

Maintain[WC-SafeDistanceBetwTrains]
Maintain[Safe Speed/AccelCom'ed]
Maintain[Safe TrainRespToComd]
OnBoard TrainControl
Mt[AccurateEstimate OfSpeed/Position]
Mt[SafeComdTo NextTrainFromEstim]
Tracking System
Achv[ComdMsg SentInTime]
Mt[Safe ComdMsg]
Achv[SentMsg DeliveredInTime]
Mt[Msg Implem]
Speed/Accel Control
Communic Infrastruct

Modeling goals & responsibilities



Modeling objects

Goal-oriented UML class diagrams



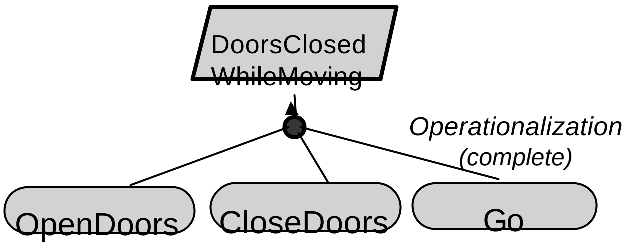
Background: goal-oriented RE (8)

- Goal operationalization:

  G is correctly operationalized by $Op_1, ..., Op_n$ iff the specs of $Op_1, ..., Op_n$ are necessary & sufficient for ensuring G

  $\{Spec(Op_1), ..., Spec(Op_n)\} \models G$ completeness

  $G \models \{Spec(Op_1), ..., Spec(Op_n)\}$ minimality

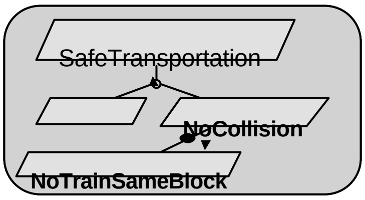


Outline

- 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

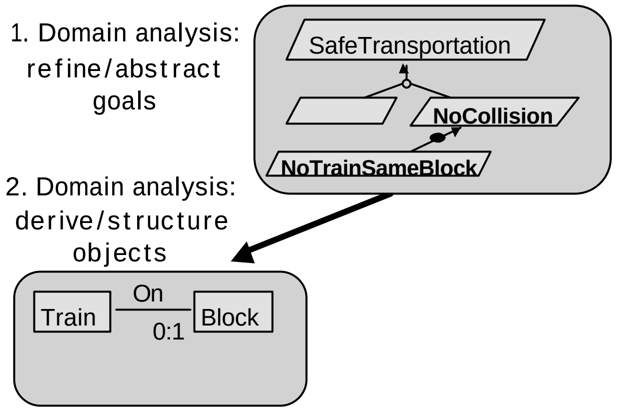
1. Domain analysis: refine/abstract goals



The KAOS goal-oriented RE method

1. Domain analysis: refine/abstract goals

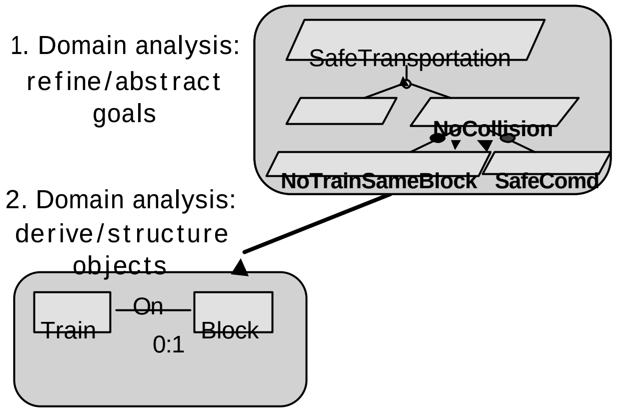
2. Domain analysis: derive/structure objects



The KAOS goal-oriented RE method

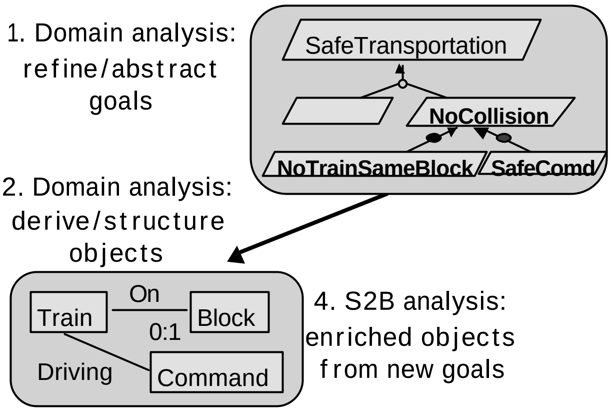
1. Domain analysis: refine/abstract goals

2. Domain analysis: derive/structure objects



3. S2B analysis: enriched goals (alternatives)

The KAOS goal-oriented RE method



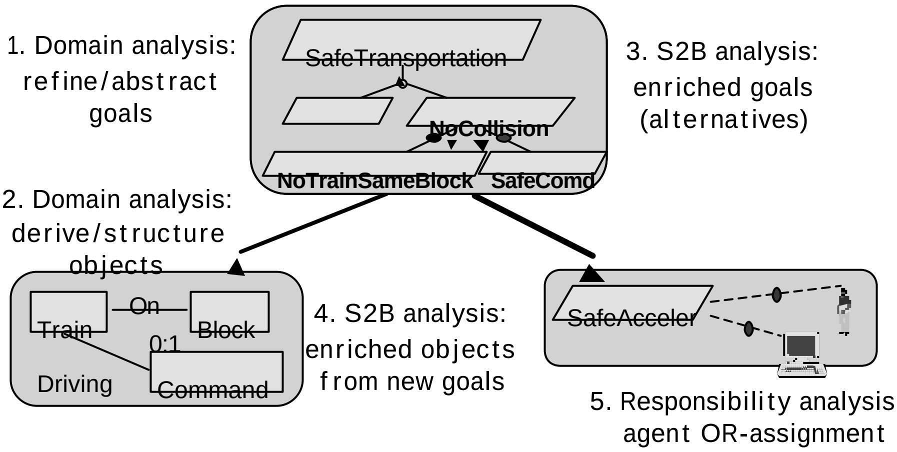
1. Domain analysis: refine/abstract goals

2. Domain analysis: derive/structure objects

3. S2B analysis: enriched goals (alternatives)

4. S2B analysis: enriched objects from new goals

The KAOS goal-oriented RE method



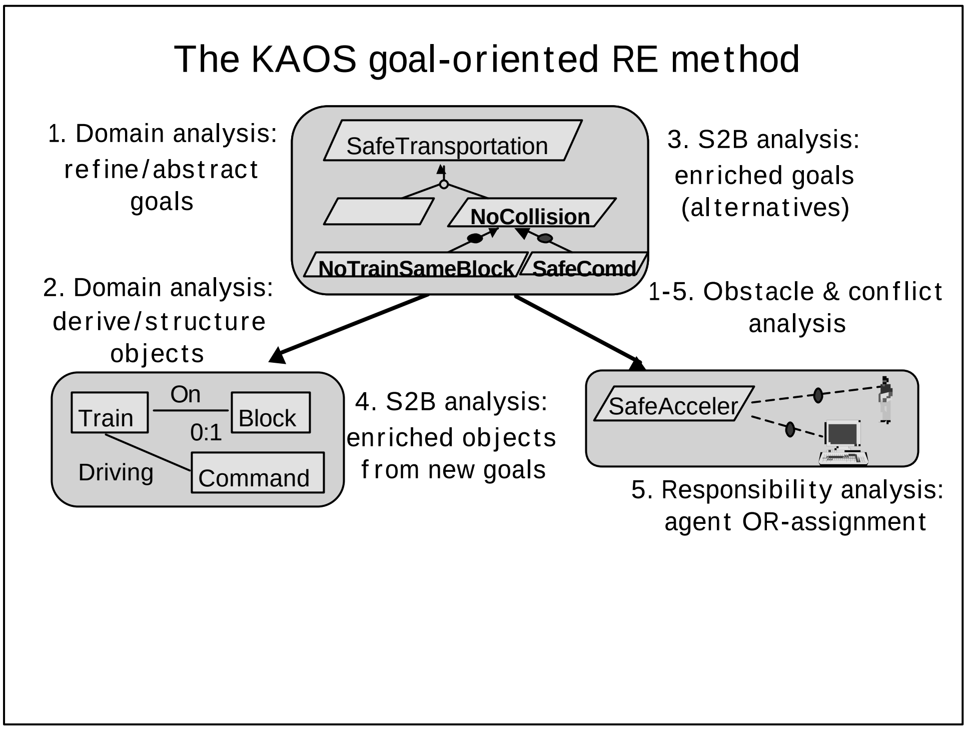
1. Domain analysis: refine/abstract goals

2. Domain analysis: derive/structure objects

3. S2B analysis: enriched goals (alternatives)

4. S2B analysis: enriched objects from new goals

5. Responsibility analysis: agent OR-assignment



The KAOS goal-oriented RE method

1. Domain analysis: refine/abstract goals

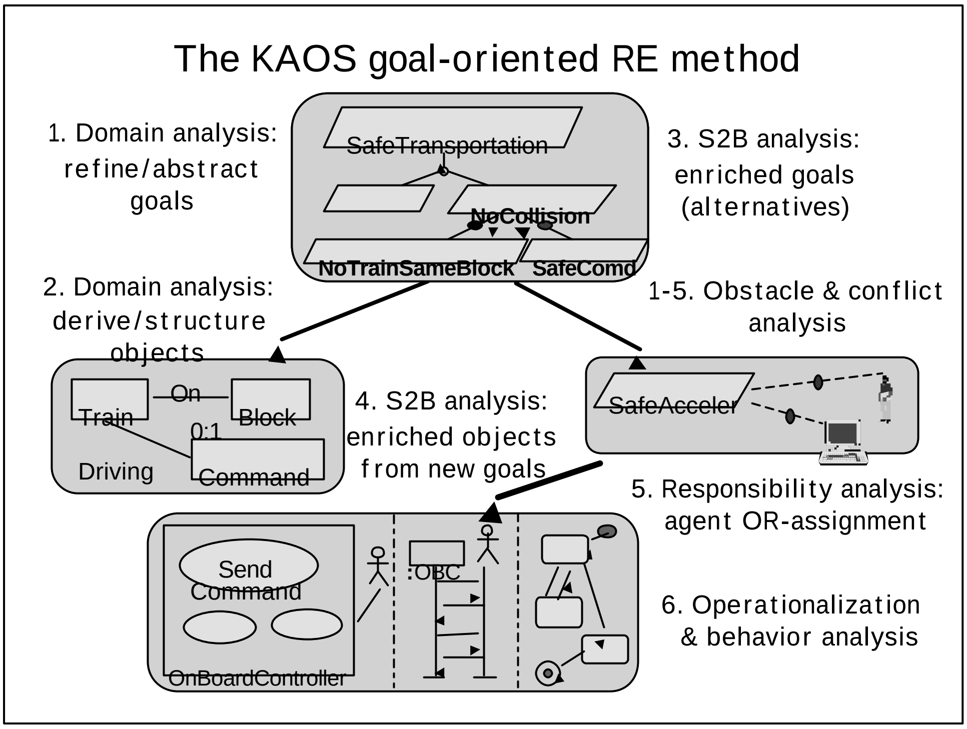
2. Domain analysis: derive/structure objects

3. S2B analysis: enriched goals (alternatives)

1-5. Obstacle & conflict analysis

4. S2B analysis: enriched objects from new goals

5. Responsibility analysis: agent OR-assignment



The KAOS goal-oriented RE method

1. Domain analysis: refine/abstract goals

2. Domain analysis: derive/structure objects

3. S2B analysis: enriched goals (alternatives)

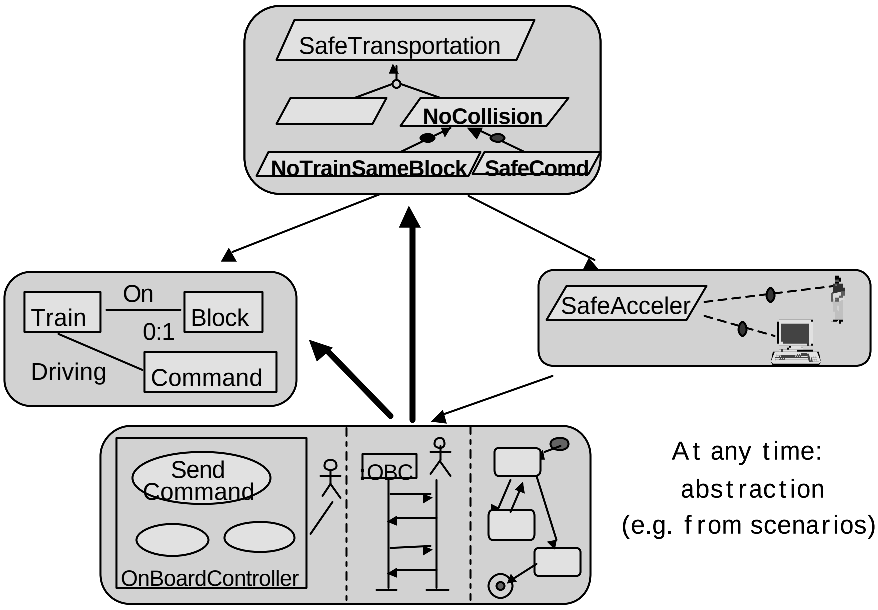
1-5. Obstacle & conflict analysis

4. S2B analysis: enriched objects from new goals

5. Responsibility analysis: agent OR-assignment

6. Operationalization & behavior analysis

The KAOS goal-oriented RE method



At any time:
abstraction
(e.g. from scenarios)

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

$\square_{\leq T}$P: P shall hold in every future state up to T time units

à$_{\leq T}$ P: P shall hold within T time units

+ past operators: "black" symbols

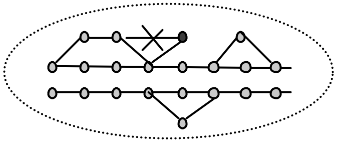
@P: ● ¬ P Ù P

Specifying goals: formal

Goal *Maintain* [DoorsClosedWhileMoving]
...
FormalDef ∀ tr: Train, s: Station
At (tr, st) Ù **o** ¬ At (tr, st) ⇒
tr.Doors = "closed" ***W*** At (tr, next(st))

Goal *Achieve* [NoDelay]
...
FormalDef ∀ tr: Train, s: Station
At (tr, st) ⇒ à$_{\leq T}$ At (tr, next(st))

characterizes maximal set of intended behaviors



Specifying operations: formal

Operation OpenDoors

Input tr: Train ; Output tr': Train

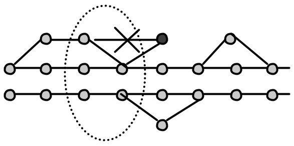
DomPre tr.Doors = "closed" domain description

DomPost tr.Doors = "open"

ReqPre for *DoorsClosedWhileMoving:* permission
∃ s: Station At (tr, s)

ReqTrig for *NoDelay:* obligation
Stopped (tr)

characterizes maximal set of intended states at snapshot

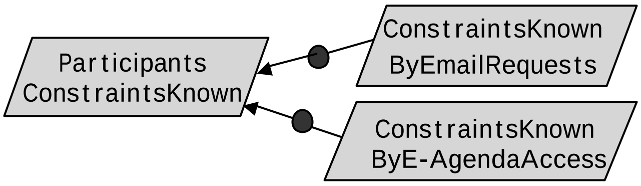


Outline

- 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

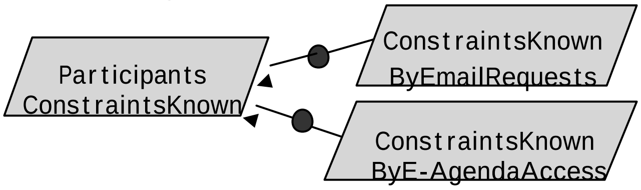
Intertwining between late RE & early AD

(1) Alternative goal refinements

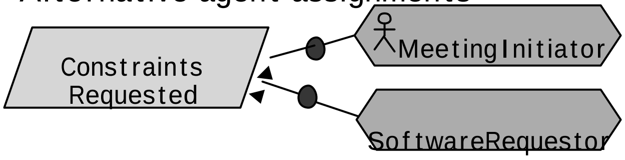


Intertwining between late RE & early AD

(1) Alternative goal refinements



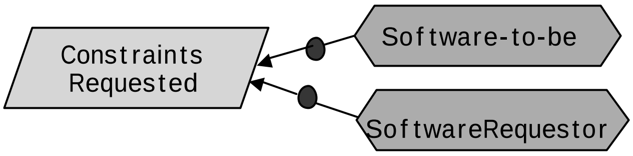
(2) Alternative agent assignments



= *early “architectural” choices to meet QoS goals*

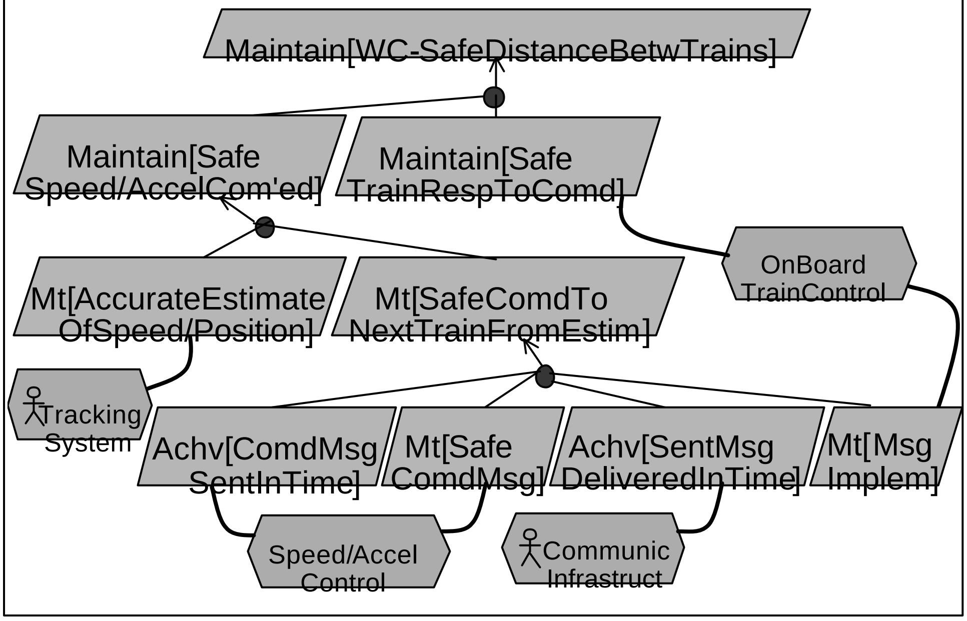
Intertwining between late RE & early AD

(3) Alternative granularities for software agents

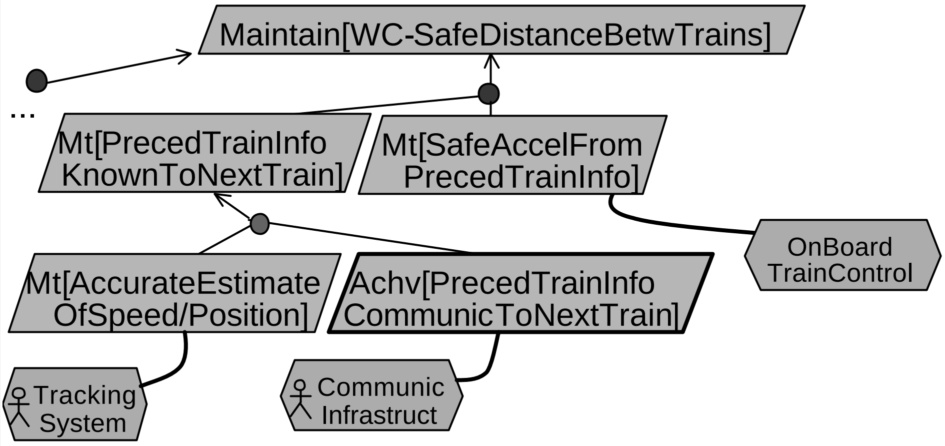


Fine, function-level granularity will be selected to meet NFR Maximize [Cohesion (C)]

Intertwining between late RE & early AD


Maintain[WC-SafeDistanceBetwTrains]
Maintain[Safe Speed/AccelCom'ed]
Maintain[Safe TrainRespToComd]
OnBoard TrainControl
Mt[AccurateEstimate OfSpeed/Position]
Mt[SafeComdTo NextTrainFromEstim]
Tracking System
Achv[ComdMsg SentInTime]
Mt[Safe ComdMsg]
Achv[SentMsg DeliveredInTime]
Mt[Msg Implem]
Speed/Accel Control
Communic Infrastruct

Alternative goal refinement & assignment


Maintain[WC-SafeDistanceBetwTrains]
...
Mt[PrecedTrainInfo KnownToNextTrain]
Mt[SafeAccelFrom PrecedTrainInfo]
OnBoard TrainControl
Mt[AccurateEstimate OfSpeed/Position]
Achv[PrecedTrainInfo CommunicToNextTrain]
Tracking System
Communic Infrastruct

different system proposal: fully distributed system

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 |-- G

  R: conjunctive set of requirements or subgoals

Checking goal refinements

- Aim: show that refinement is correct & complete

  R, Ass, Dom |-- 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 |-- 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:

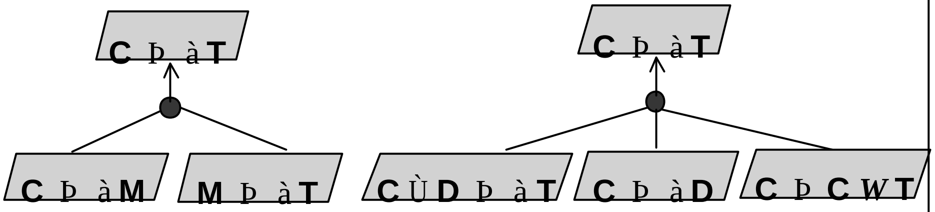
  - to complete partial refinements
  - to explore alternative refinements

Checking goal refinements (2)

Idea:

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

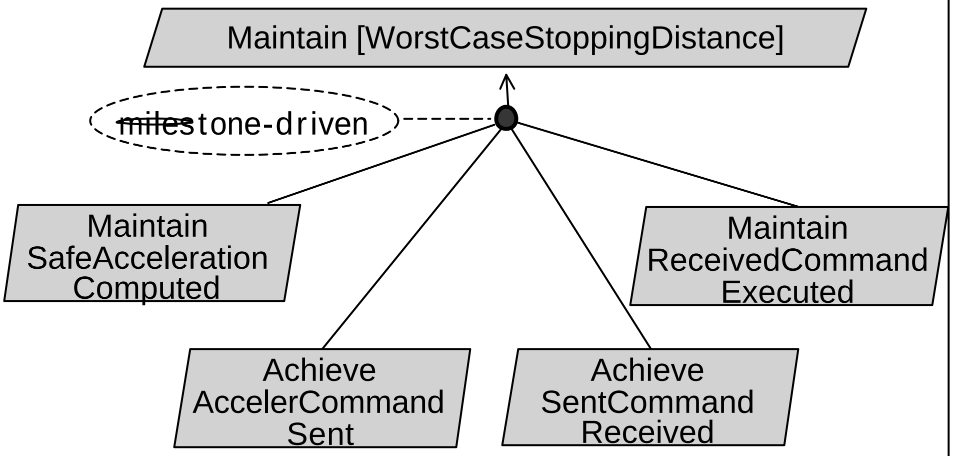
e.g. frequent patterns:



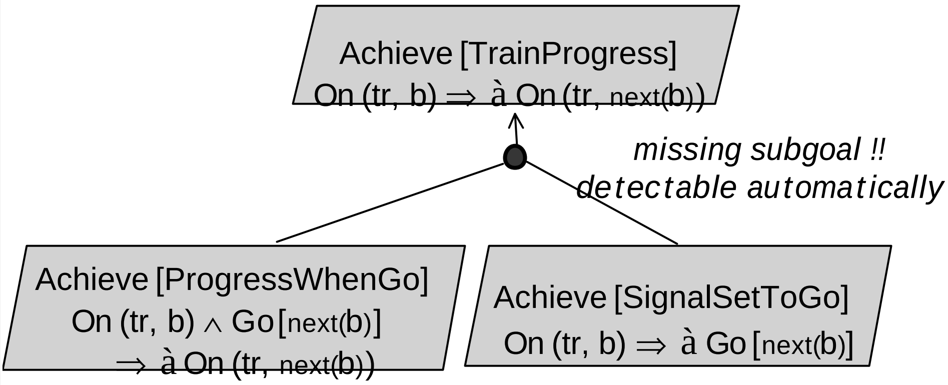
milestone-driven

case-driven

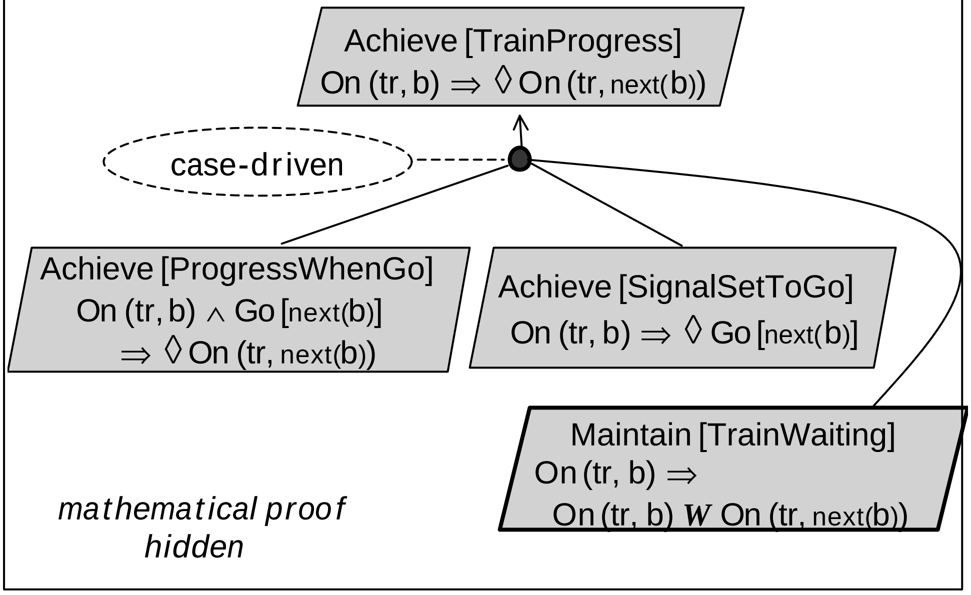
Checking goal refinements (3)


Maintain [WorstCaseStoppingDistance]
~~miles~~tone-driven
Maintain SafeAcceleration Computed
Maintain ReceivedCommand Executed
Achieve AccelerCommand Sent
Achieve SentCommand Received

Checking goal refinements (4)



Checking goal refinements (4)


Achieve [TrainProgress]
On (tr, b) ⇒ ◊ On (tr, next(b))
case-driven
Achieve [ProgressWhenGo]
On (tr, b) ∧ Go [next(b)] ⇒ ◊ On (tr, next(b))
Achieve [SignalSetToGo]
On (tr, b) ⇒ ◊ Go [next(b)]
Maintain [TrainWaiting]
On (tr, b) ⇒ On (tr, b) W On (tr, next(b))

mathematical proof hidden

Checking goal refinements (5)

- Approach 3: Early bounded model checking
  - checking of goal models
  - partial models
  - incremental checking/debugging
  - on selected object instances *(propositionalization)*
  - ouput:

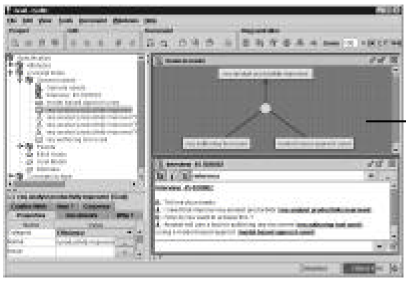
    OK

    KO + counter-example scenario

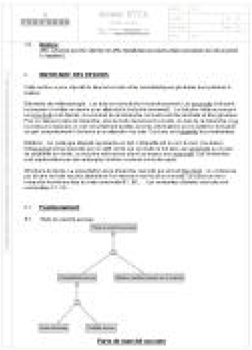
Roundtrip use of SAT solver, NuSMV, theorem prover

Time for demo...

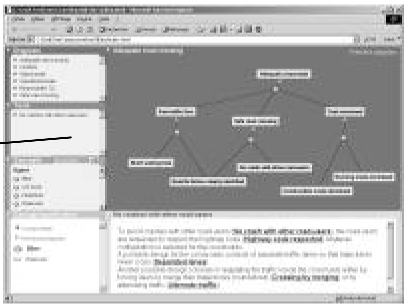
The GRAIL tool



KAOS model editor

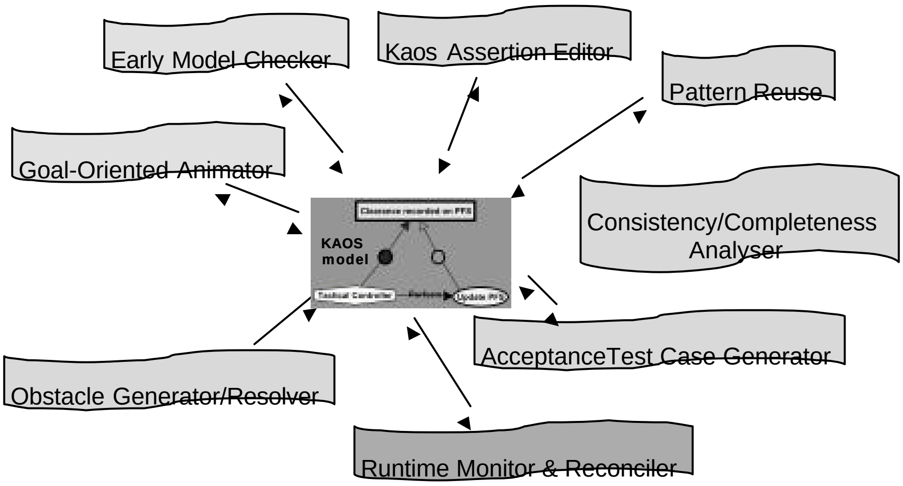


Requirements documents generation



KAOS model browser

The GRAIL/FAUST toolkit



Early Model Checker

Kaos Assertion Editor

Pattern Reuse

Goal-Oriented Animator

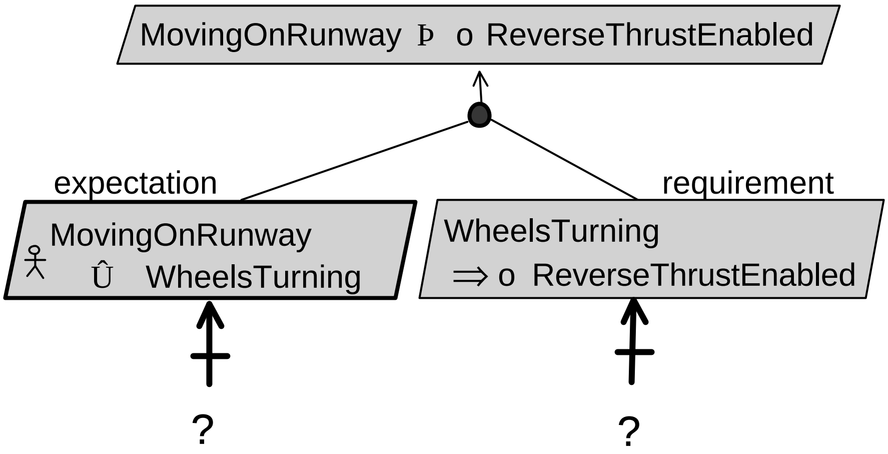
Consistency/Completeness Analyser

AcceptanceTest Case Generator

Obstacle Generator/Resolver

Runtime Monitor & Reconciler

Generating obstacles



MovingOnRunway Þ o ReverseThrustEnabled

expectation

MovingOnRunway
Û WheelsTurning

requirement

WheelsTurning
⇒ o ReverseThrustEnabled

?

?

Generating obstacles (2)

- Deriving precondition for obstruction

MovingOnRunway Þ WheelsTurning

Generating obstacles (2)

◆ Deriving precondition for obstruction

MovingOnRunway Þ WheelsTurning

® goal negation:

à MovingOnRunway Ù ¬ WheelsTurning

Generating obstacles (2)

◆ Deriving precondition for obstruction

MovingOnRunway Þ WheelsTurning

® goal negation:

à MovingOnRunway Ù ¬ WheelsTurning

® regress through Dom:

? necessary conditions for wheels turning ?

WheelsTurning Þ ¬ Aquaplaning

i.e. Aquaplaning Þ ¬ WheelsTurning

Generating obstacles (2)

◆ Deriving precondition for obstruction

MovingOnRunway Þ WheelsTurning

® goal negation:

à MovingOnRunway Ù ¬ WheelsTurning

® regress through Dom:

? necessary conditions for wheels turning ?

WheelsTurning Þ ¬ Aquaplaning

i.e. Aquaplaning Þ ¬ WheelsTurning

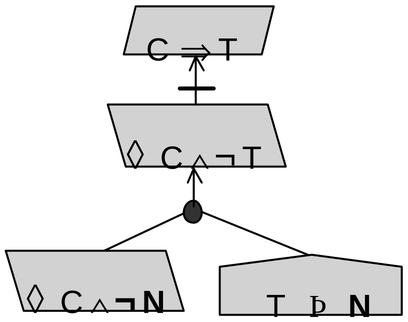
® RHS unifiable:

à MovingOnRunway Ù Aquaplaning *Warsaw obstacle*

Generating obstacles (3)

◆ Using formal obstruction patterns

in fact we just used a frequent pattern:



C ⇒ T

◊ C ∧ ¬T

◊ C ∧ ¬N

T Þ N

obstacle *domain property*

Verifying/deriving operationalizations

- Build a library of formal operationalization patterns for frequent goal specification patterns

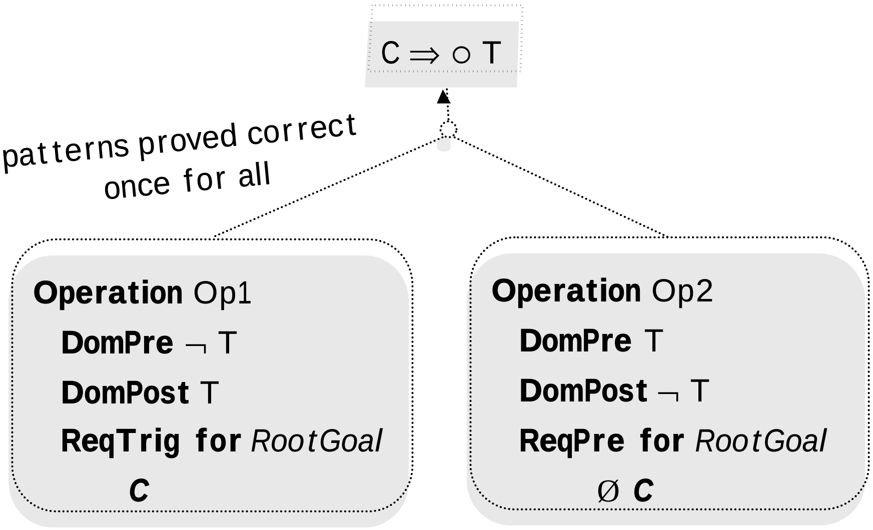
  e.g. Achieve goals: $C \Rightarrow à_{\leq d}\ T$ $C \Rightarrow \circ\ T$

  Maintain goals: $C \Rightarrow \square\ T$ $C \Rightarrow T\ \boldsymbol{W}\ N$

  + extensions adapted from Dwyer et al

- Prove pattern correctness once for all
- Reuse through instantiation, in matching situations

Verifying/deriving operationalizations



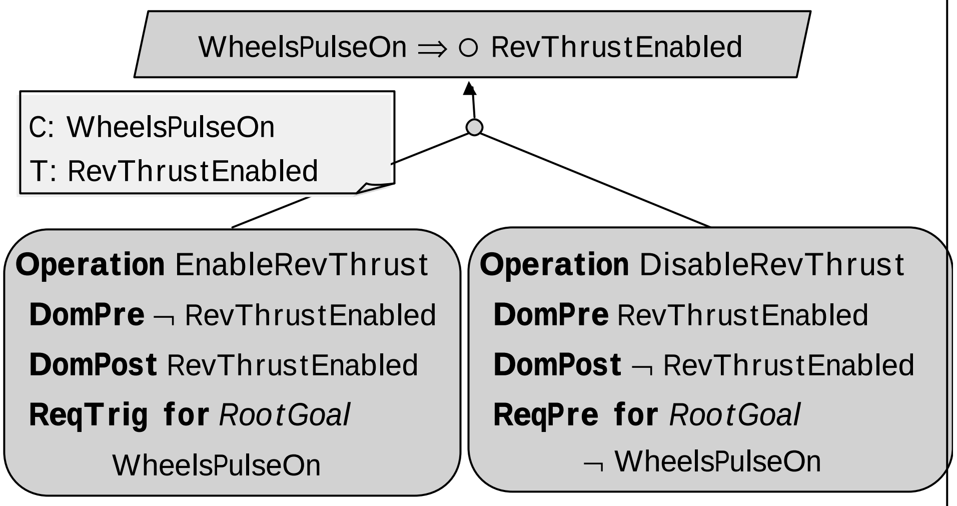
$C \Rightarrow \circ\ T$

patterns proved correct
once for all

Operation Op1
DomPre ¬ T
DomPost T
ReqTrig for *RootGoal*
C

Operation Op2
DomPre T
DomPost ¬ T
ReqPre for *RootGoal*
Ø ***C***

Verifying/deriving operationalizations



$\text{WheelsIsPulseOn} \Rightarrow \circ\ \text{RevThrustEnabled}$

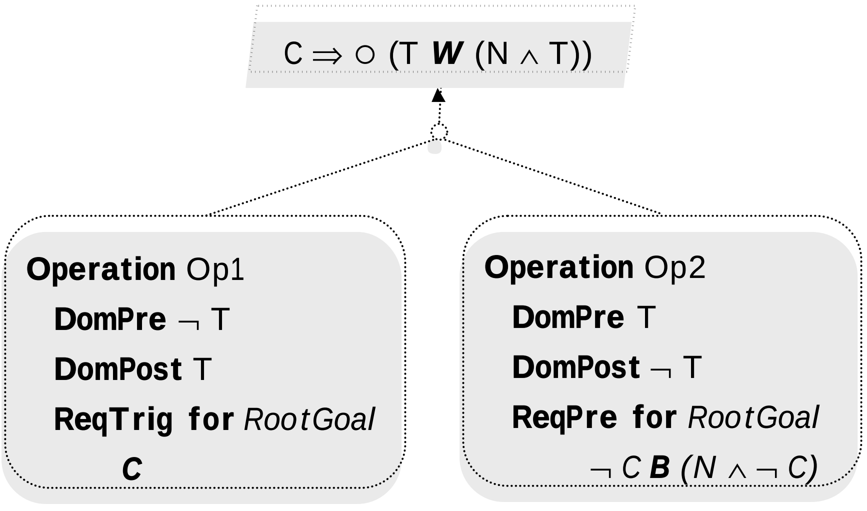
C: WheelsIsPulseOn
T: RevThrustEnabled

Operation EnableRevThrust
DomPre $\neg$ RevThrustEnabled
DomPost RevThrustEnabled
ReqTrig for *RootGoal*
WheelsIsPulseOn

Operation DisableRevThrust
DomPre RevThrustEnabled
DomPost $\neg$ RevThrustEnabled
ReqPre for *RootGoal*
$\neg$ WheelsIsPulseOn

Verifying/deriving operationalizations

"T shall hold between C and N"

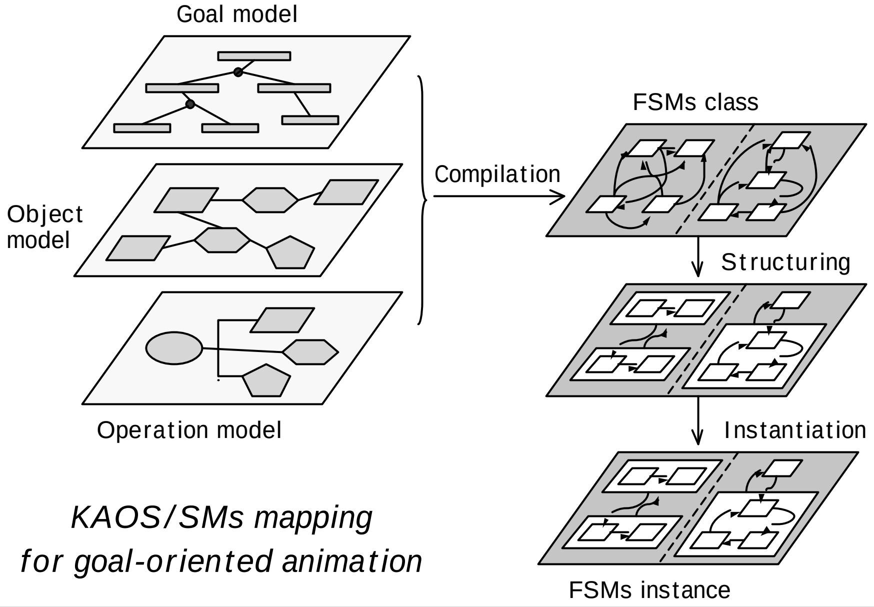


$C \Rightarrow \circ\ (T\ \boldsymbol{W}\ (N \wedge T))$

Operation Op1
DomPre $\neg$ T
DomPost T
ReqTrig for *RootGoal*
C

Operation Op2
DomPre T
DomPost $\neg$ T
ReqPre for *RootGoal*
$\neg C\ \boldsymbol{B}\ (N \wedge \neg C)$

Generating state machines from goal operationalizations



KAOS/SMs mapping
for goal-oriented animation

Generating state machines from goal operationalizations (2)

Step 1: Build FSM class declarations

```
for each e: Entity ∪ 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 DomPre/DomPost
        identify additional legal states of attr in Goal
```

Goal Maintain[DoorsClosedWhileMoving]
  FormaDef ∀tr: Train, s: Station
   At (tr, s) ∧ ○ ¬ At (tr, s)
    ⇒ tr.Doors = ‘closed’ ***W*** At (tr, next(s))

Entity Station

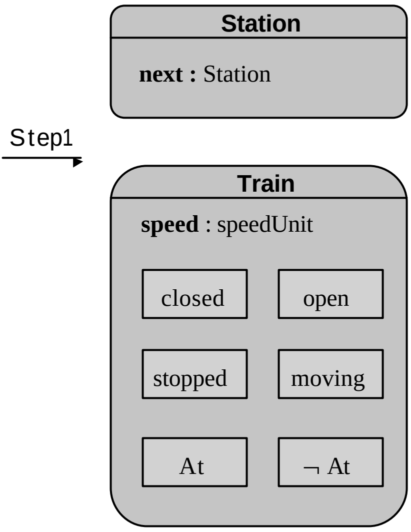
Entity Train
  Has Speed: speedUnit

Relationship next
  Links Station {card 0:1}

Relationship At
  Links Train {card 0:1}, Station {card 0:N}

Operation OpenDoors
  Input tr: Train; **Output tr:** Train
  DomPre tr.Doors = ‘closed’
  DomPost tr.Doors = ‘open’
  ReqPre for DoorsClosedWhileMoving
    ∃ s : Station At (tr,s)

Operation StartTrain
  Input tr: Train; **Output tr:** Train
  DomPre tr.Status = ‘stopped’
  DomPost tr.Status = ‘moving’
  ReqPre for DoorsClosedWhileMoving
    tr.Doors = ‘closed’



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;

Goal Maintain[DoorsClosedWhileMoving]
FormaDef ∀tr: Train, s: Station
At (tr, s) ∧ ❍ ¬ At (tr, s)
⇒ tr.Doors = ‘closed’ ***W*** At (tr, next(s))

Entity Station

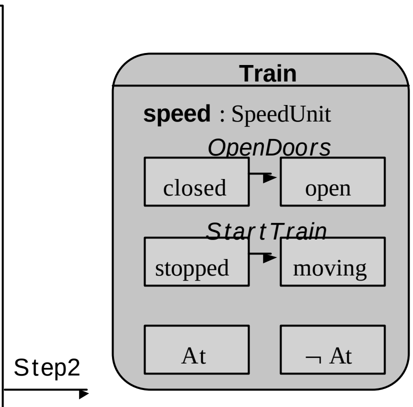
Entity Train
Has speed: SpeedUnit

Relationship next
Links Station {card 0:1}

Relationship At
Links Train {card 0:1}, Station {card 0:N}

Operation OpenDoors
Input tr: Train; **Output** tr: Train
DomPre tr.Doors = ‘closed’
DomPost tr.Doors = ‘open’
ReqPre for DoorsClosedWhileMoving:
∃ s : Station At (tr, s)

Operation StartTrain
Input tr: Train**;** **Output** tr: Train
DomPre tr.status = ‘stopped’
DomPost tr.status = ‘moving’
ReqPre for DoorsClosedWhileMoving:
tr.Doors = ‘closed’

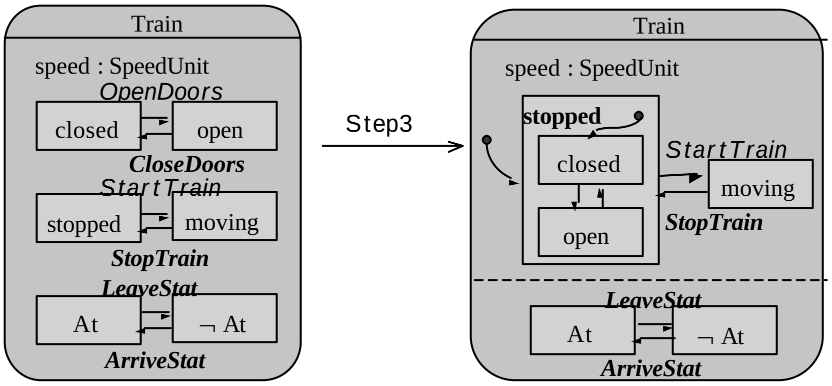


Transition StartTrain
SourceState stopped
DestState moving
Guard closed
Actions null

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



Transition StartTrain
SourceState stopped
DestState moving
Guard **closed**
Actions null

Transition StartTrain
SourceState stopped
DestState moving
Guard
Actions null

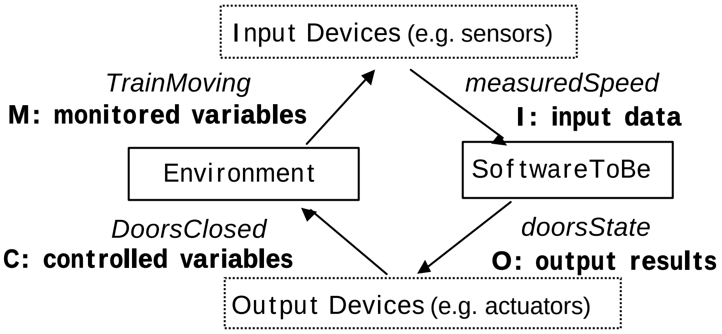
(Tran Van & AvL, 2003)

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

- Requirements vs. software specifications:

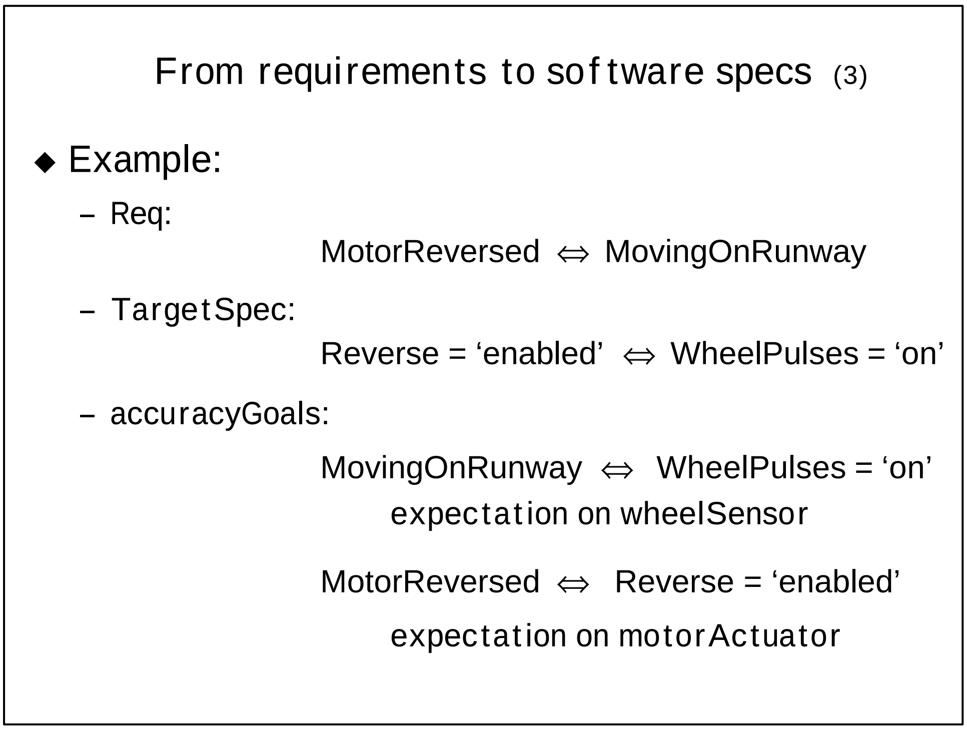


$\text{Req} \subseteq M \times C$ Spec = *Translation* (Req) such that

$\text{Spec} \subseteq I \times O$ $\{\text{Spec}, \text{Dom}\} \models \text{Req}$

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 vatiables, 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)

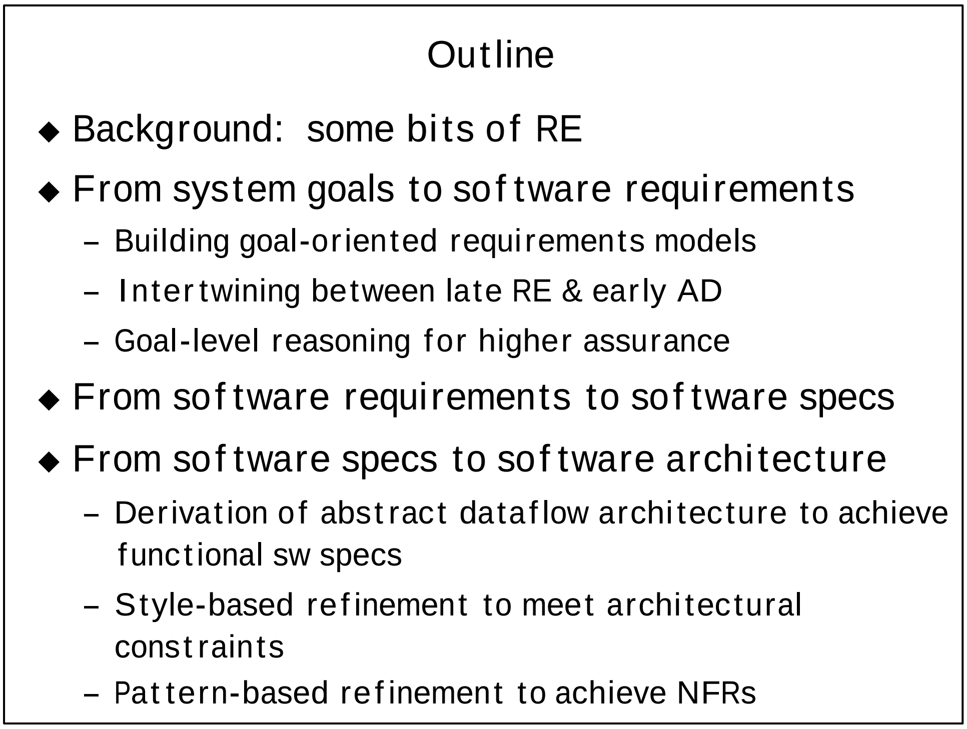
- Example:
  - Req:

    MotorReversed ⇔ MovingOnRunway
  - TargetSpec:

    Reverse = 'enabled' ⇔ WheelPulses = 'on'
  - accuracyGoals:

    MovingOnRunway ⇔ WheelPulses = 'on'
    expectation on wheelSensor

    MotorReversed ⇔ 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

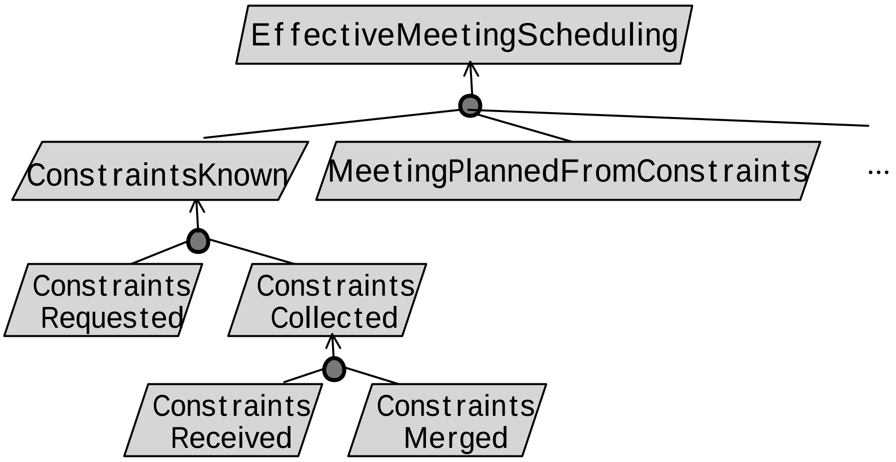
Deriving an abstract dataflow architecture

- 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*) ≡ Controls (*C1*, d) ∧ Monitors (*C2*, d)

Deriving dataflow architecture: example



EffectiveMeetingScheduling

ConstraintsKnown

MeetingPlannedFromConstraints

...

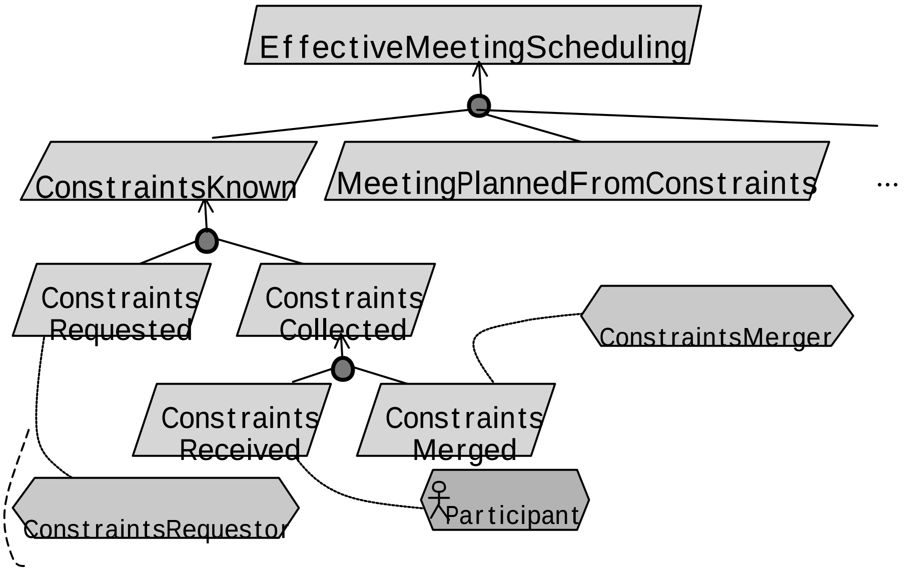
Constraints Requested

Constraints Collected

Constraints Received

Constraints Merged

Deriving dataflow architecture: example



EffectiveMeetingScheduling

ConstraintsKnown

MeetingPlannedFromConstraints

...

Constraints Requested

Constraints Collected

ConstraintsMerger

Constraints Received

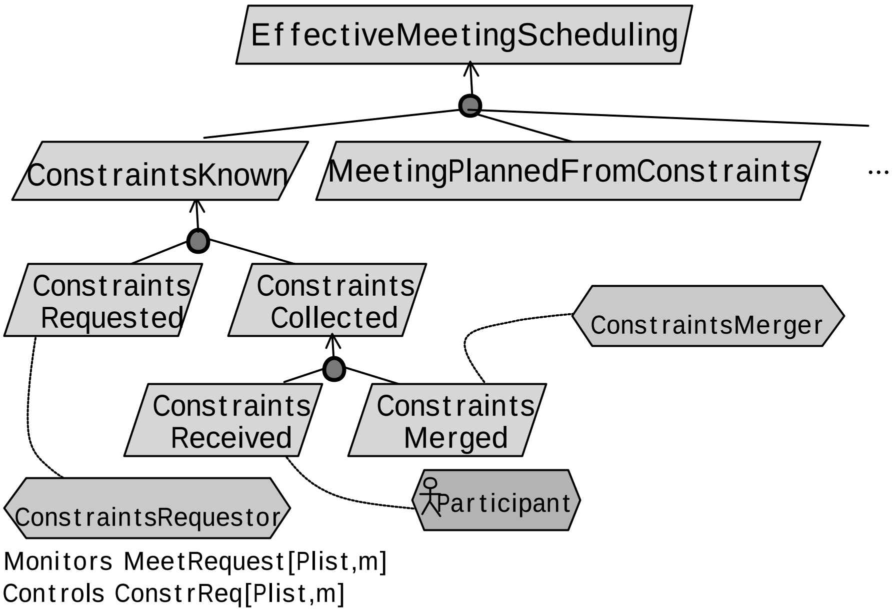
Constraints Merged

ConstraintsRequestor

Participant

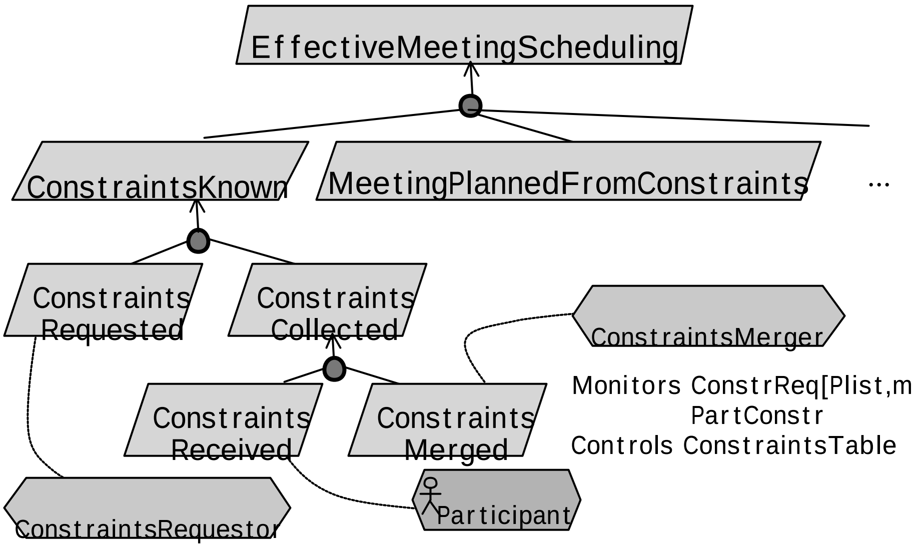
MeetRequest (Plist, m) ∧ p in Plist ⇒ à≤ $_{RT}$ ConstrReq (p, m)

Deriving dataflow architecture: example



Monitors MeetRequest[Plist,m]
Controls ConstrReq[Plist,m]

Deriving dataflow architecture: example

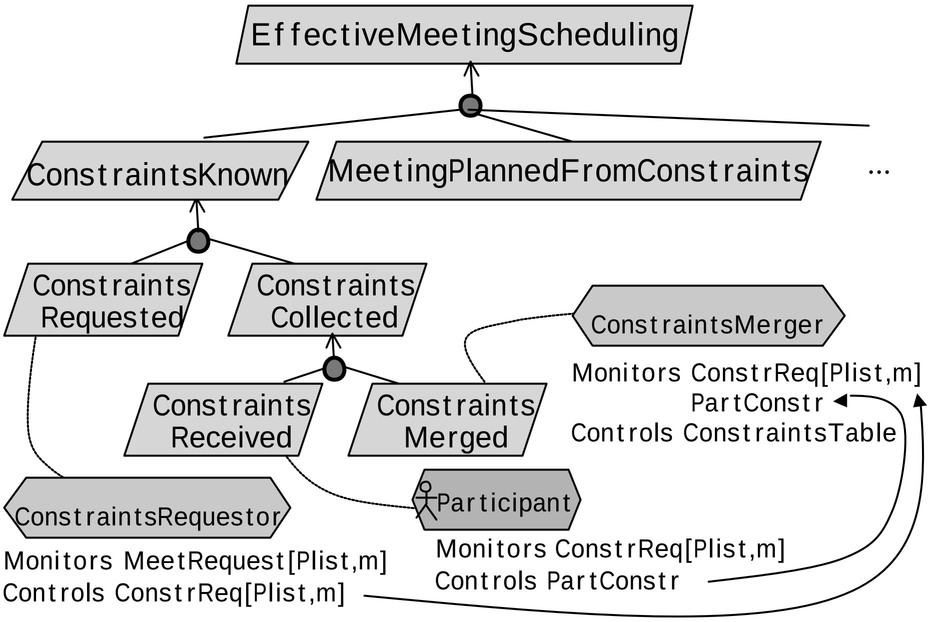


Monitors ConstrReq[Plist,m]
PartConstr
Controls ConstraintsTable

Monitors MeetRequest[Plist,m]
Controls ConstrReq[Plist,m]

Monitors ConstrReq[Plist,m]
Controls PartConstr

Deriving dataflow architecture: example

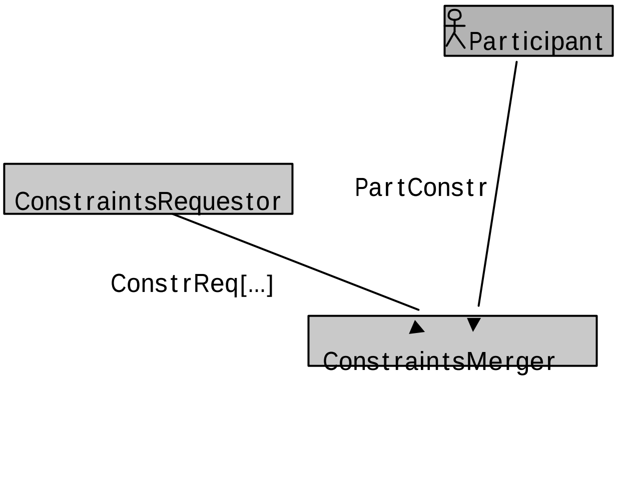


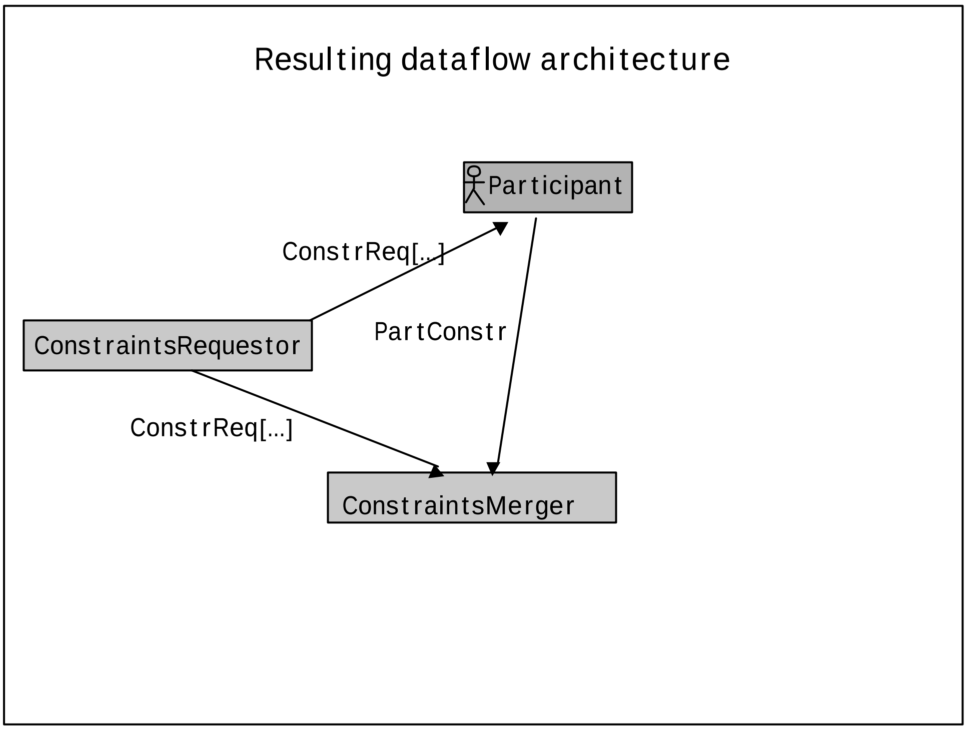
ConstraintsMerger: Monitors ConstrReq[Plist,m] PartConstr; Controls ConstraintsTable

ConstraintsRequestor: Monitors MeetRequest[Plist,m]; Controls ConstrReq[Plist,m]

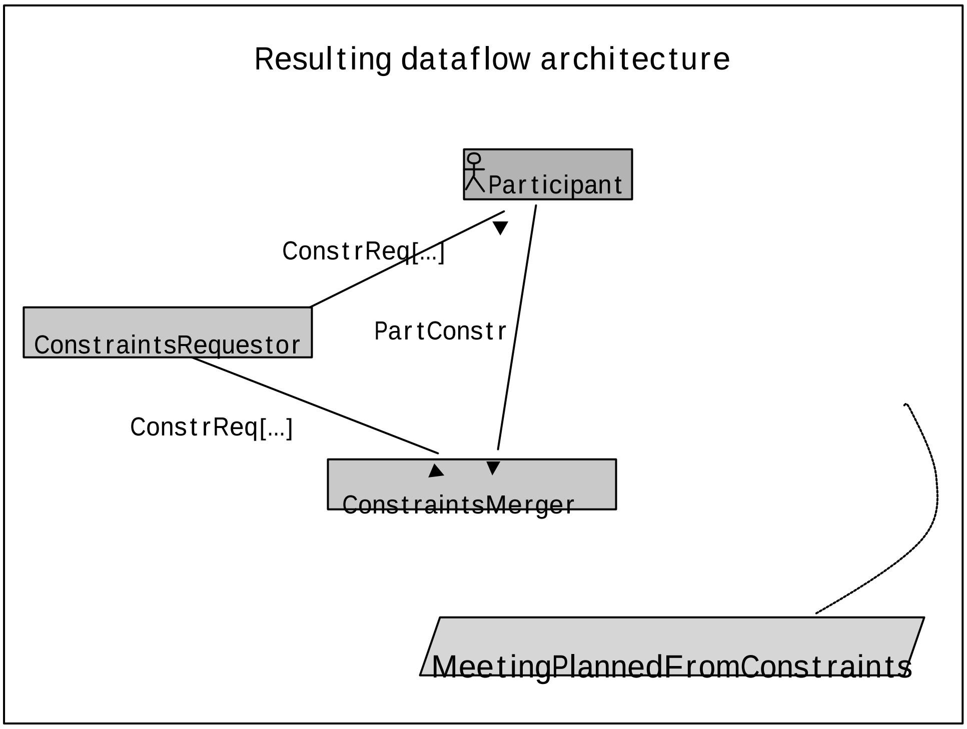
Participant: Monitors ConstrReq[Plist,m]; Controls PartConstr

Resulting dataflow architecture



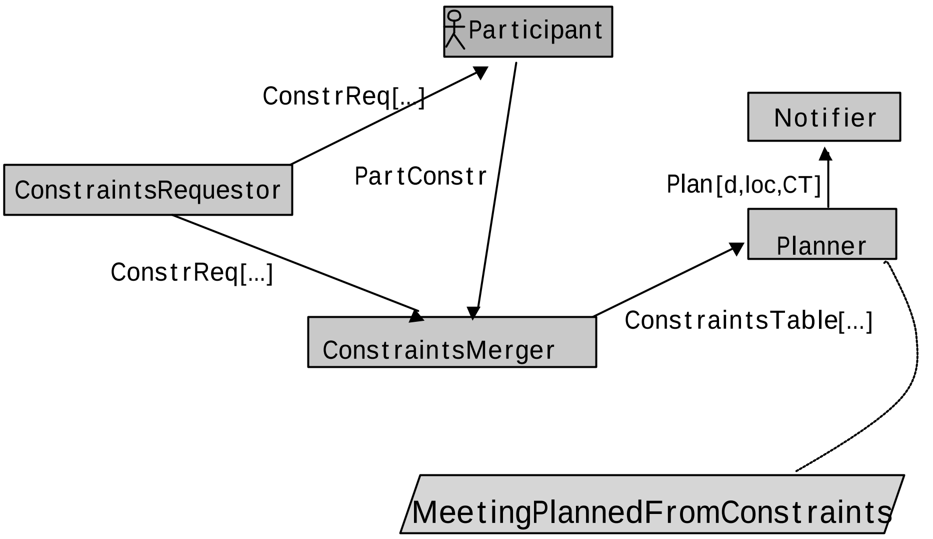


Resulting dataflow architecture

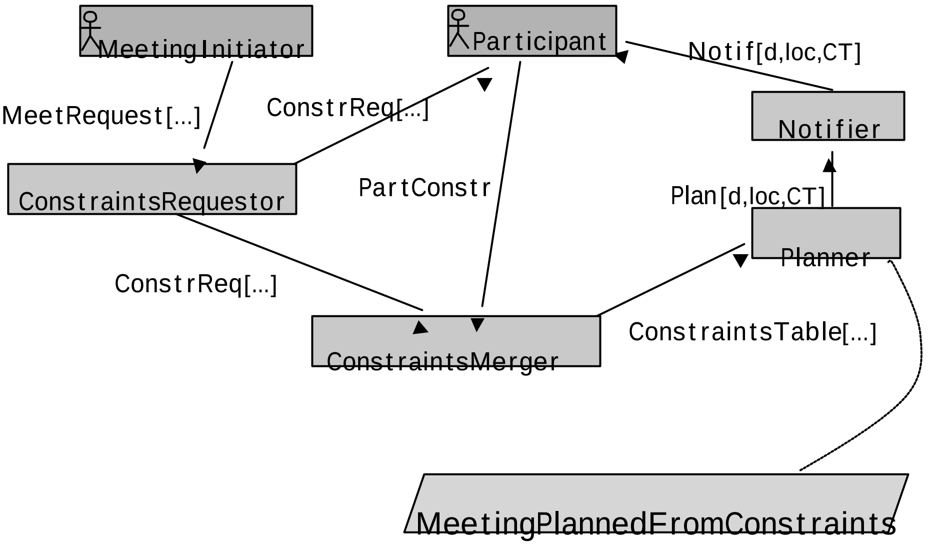


Resulting dataflow architecture

Resulting dataflow architecture



Resulting dataflow architecture



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 ...

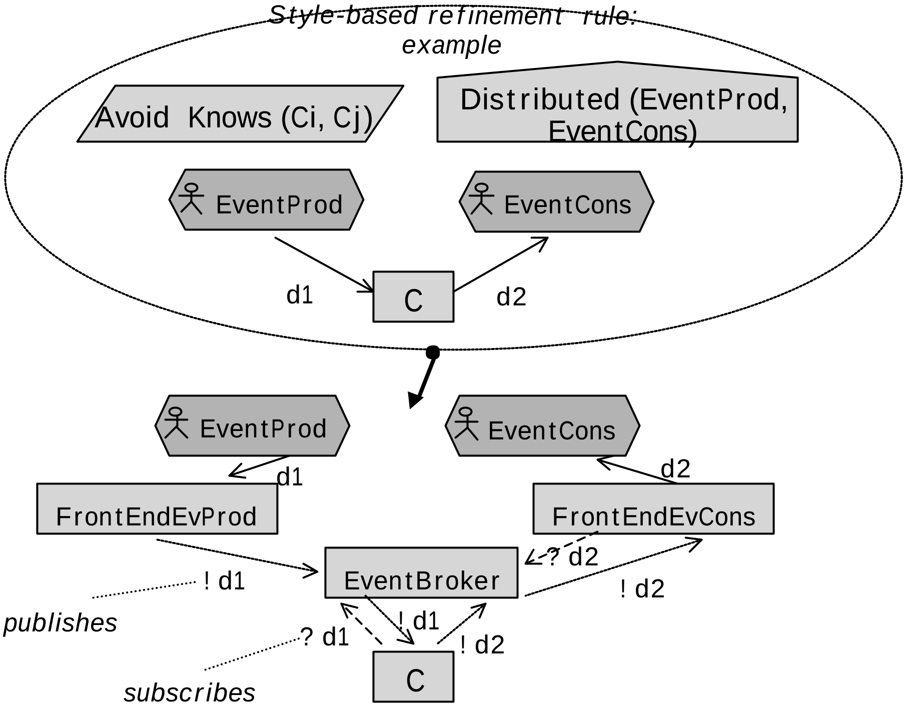
- from environment agents: features, inter-relationships
- global constraints on architectural design space

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)

Style-based refinement rule: example



Avoid Knows (Ci, Cj)

Distributed (EventProd, EventCons)

EventProd

EventCons

d1

C

d2

EventProd

EventCons

d1

d2

FrontEndEvProd

FrontEndEvCons

EventBroker

! d1

? d2

! d2

publishes

? d1

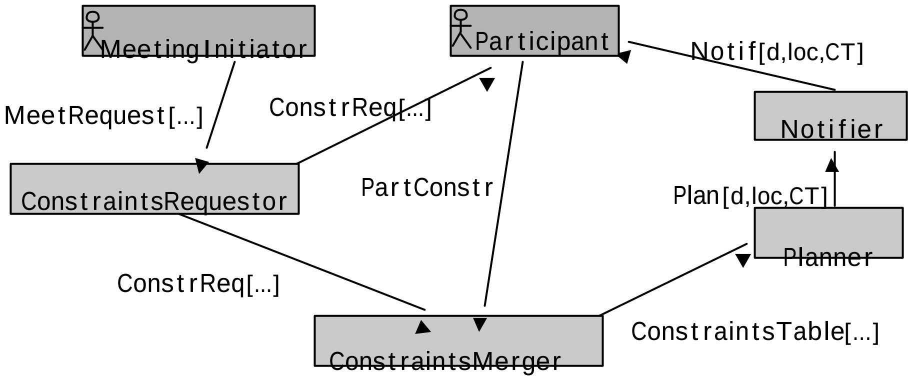
! d1

! d2

C

subscribes

From MeetingScheduler dataflow architecture ...



MeetingInitiator

Participant

Notif[d,loc,CT]

MeetRequest[...]

ConstrReq[...]

Notifier

ConstraintsRequestor

PartConstr

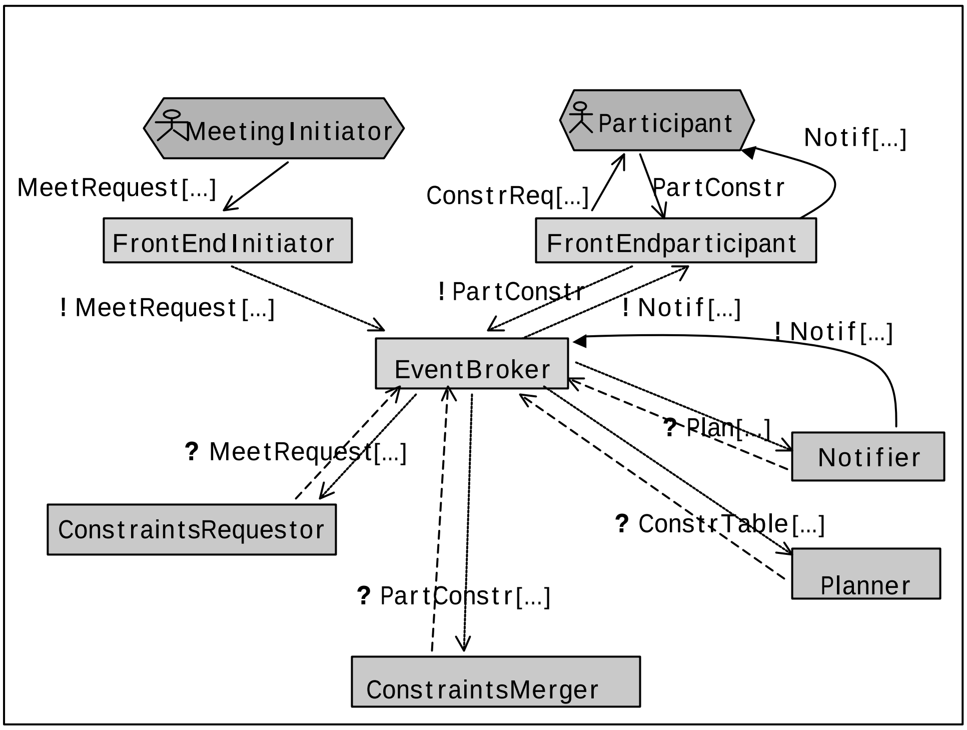
Plan[d,loc,CT]

Planner

ConstrReq[...]

ConstraintsMerger

ConstraintsTable[...]



Outline

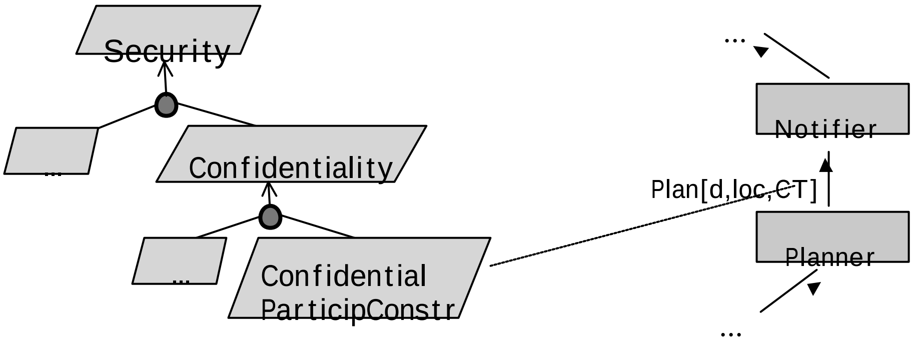
- ◆ Background: some bits of RE
- ◆ From system goals to software requirements
  - – Building goal-oriented requirements models
  - – Intertwining between late RE & early AD
  - – Goal-level reasoning for higher assurance
- ◆ From software requirements to software specs
- ◆ From software specs to software 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 contraints on single component
  - MaxCohesion (C): fine-grained functionality

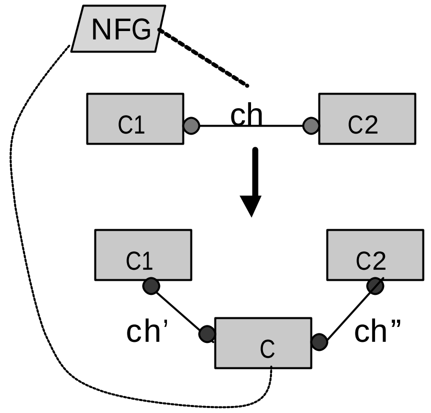
Architecture refinement (2)

1. For each terminal NFG in goal refinement graph ...
   - identify all connectors/components constrained by it
   - instantiate it to those connectors/components

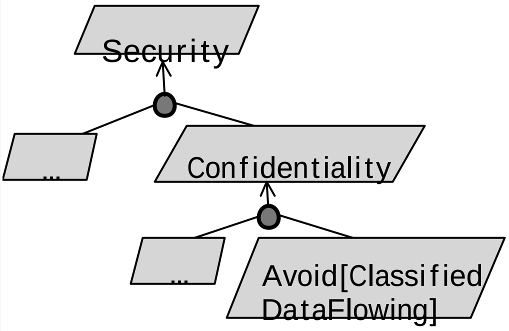


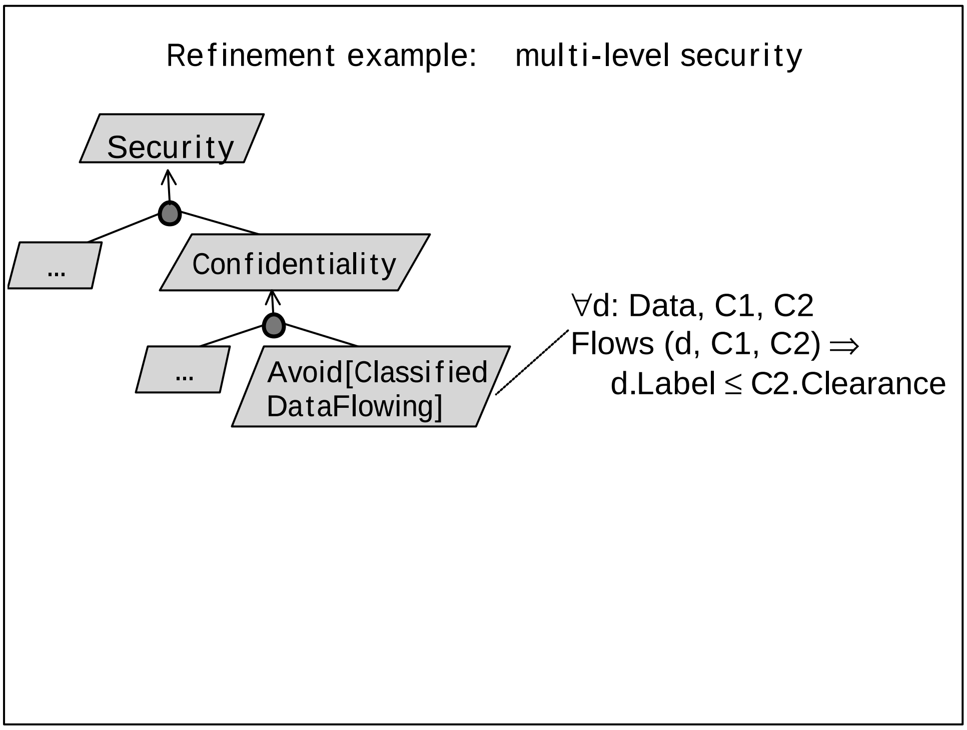
Architecture refinement (3)

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



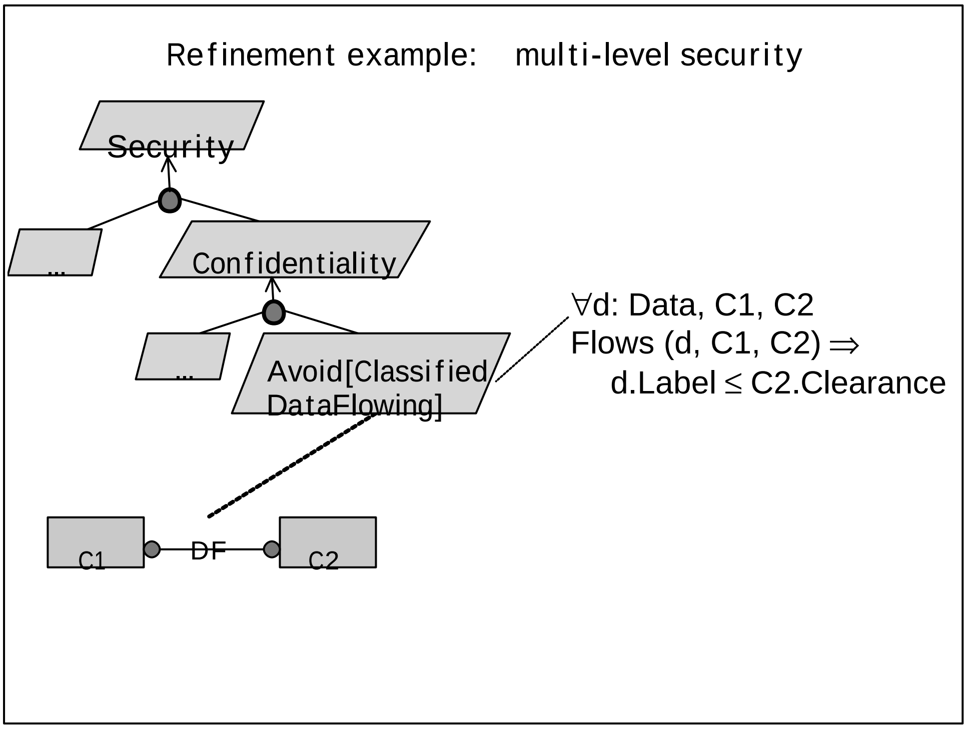
Refinement example: multi-level security





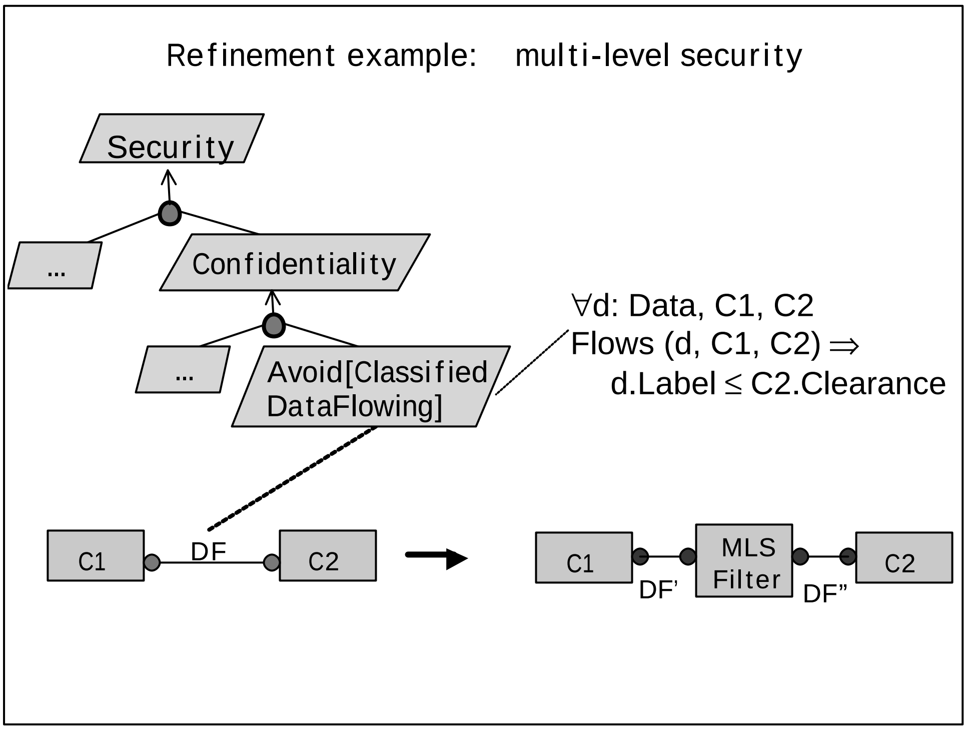
Refinement example: multi-level security

$\forall$d: Data, C1, C2
Flows (d, C1, C2) $\Rightarrow$
d.Label $\leq$ C2.Clearance



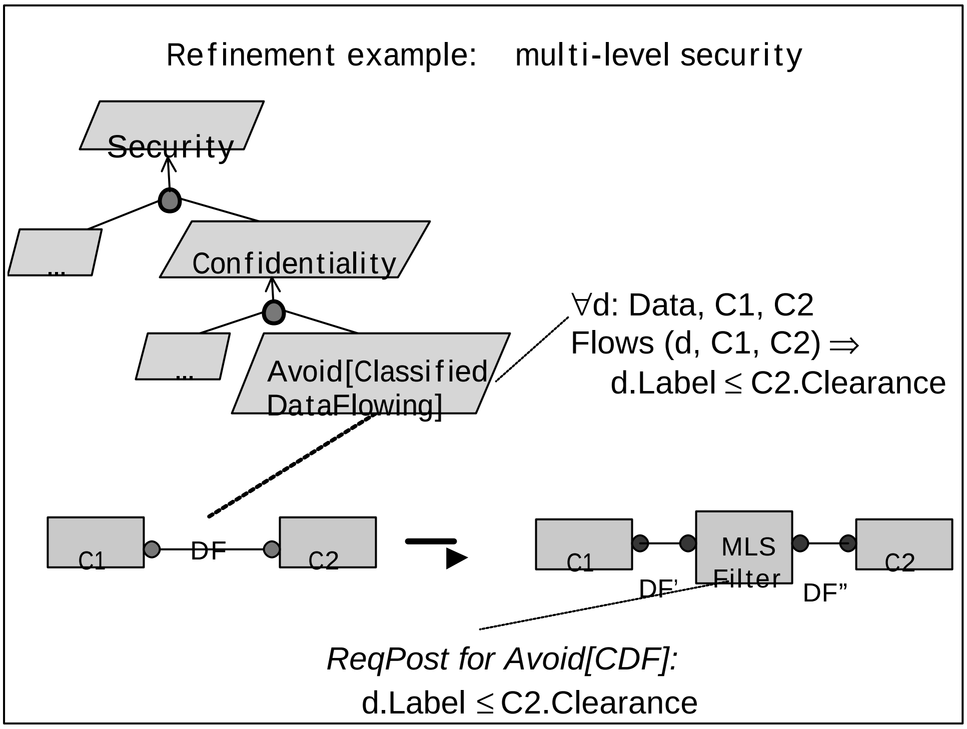
Refinement example: multi-level security

$\forall$d: Data, C1, C2
Flows (d, C1, C2) $\Rightarrow$
d.Label $\leq$ C2.Clearance



Refinement example: multi-level security

∀d: Data, C1, C2
Flows (d, C1, C2) ⇒
d.Label ≤ C2.Clearance



Refinement example: multi-level security

∀d: Data, C1, C2
Flows (d, C1, C2) ⇒
d.Label ≤ C2.Clearance

ReqPost for Avoid[CDF]:
d.Label ≤ C2.Clearance

Architecture refinement (4)

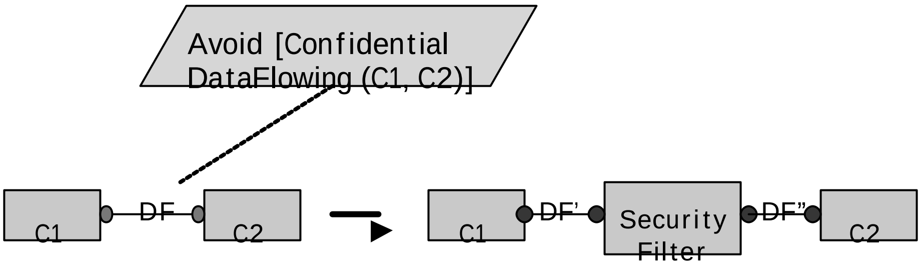
2. For each NFG-constrained connector/component ...
   - refine it to meet instantiated NFG ...

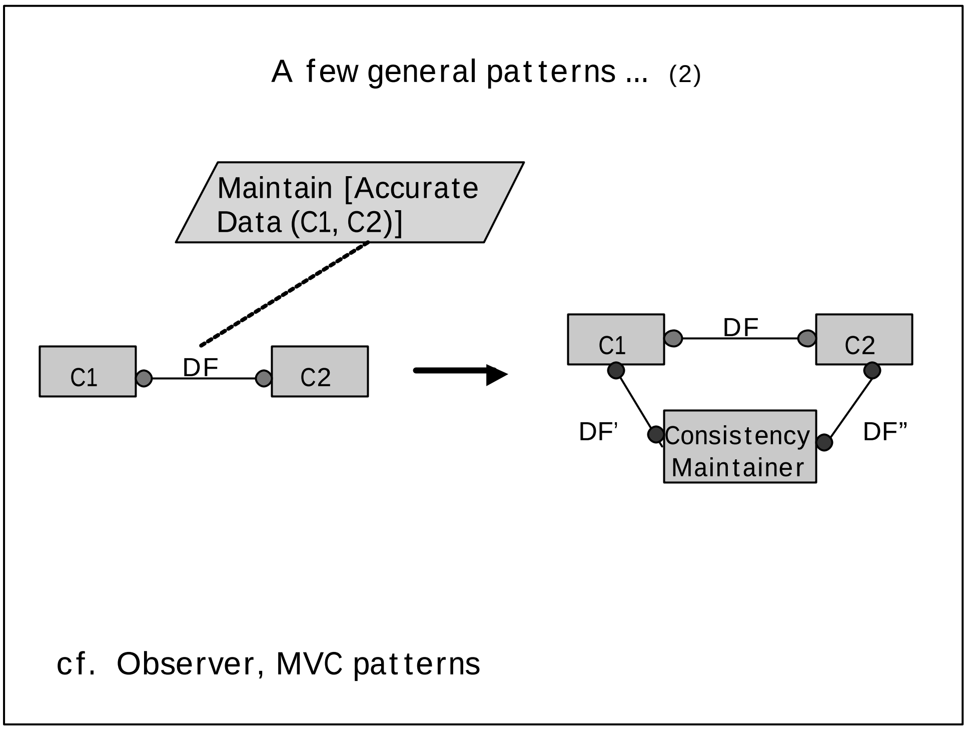
     by use of architectural refinement patterns:

     - 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)

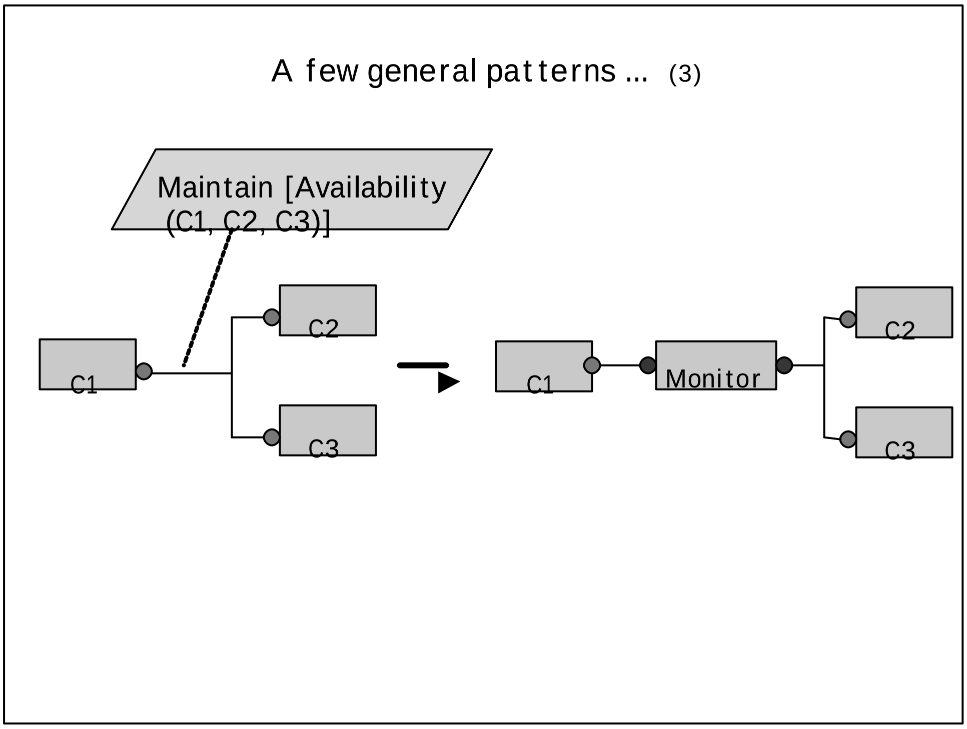
A few general patterns ...





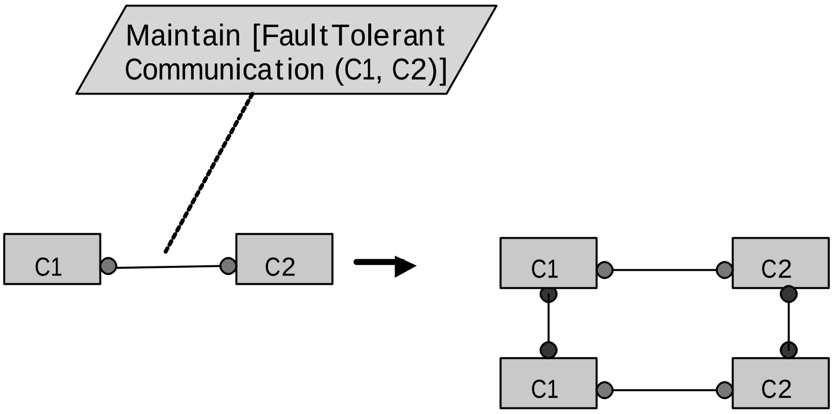
A few general patterns … (2)

cf. Observer, MVC patterns

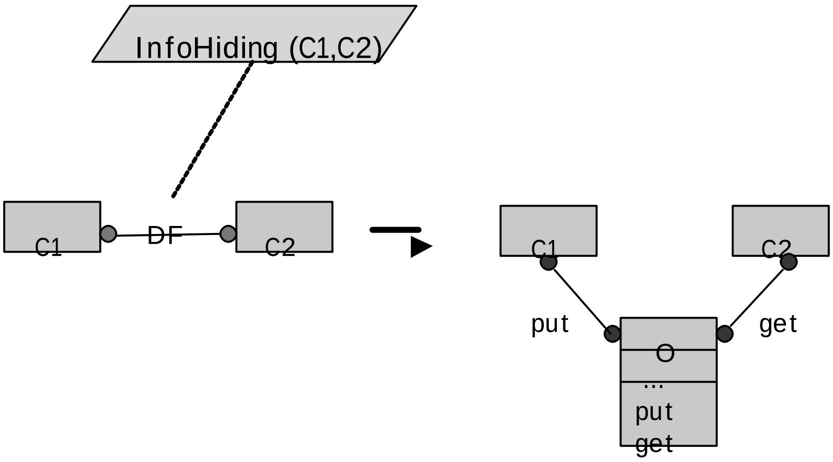


A few general patterns … (3)

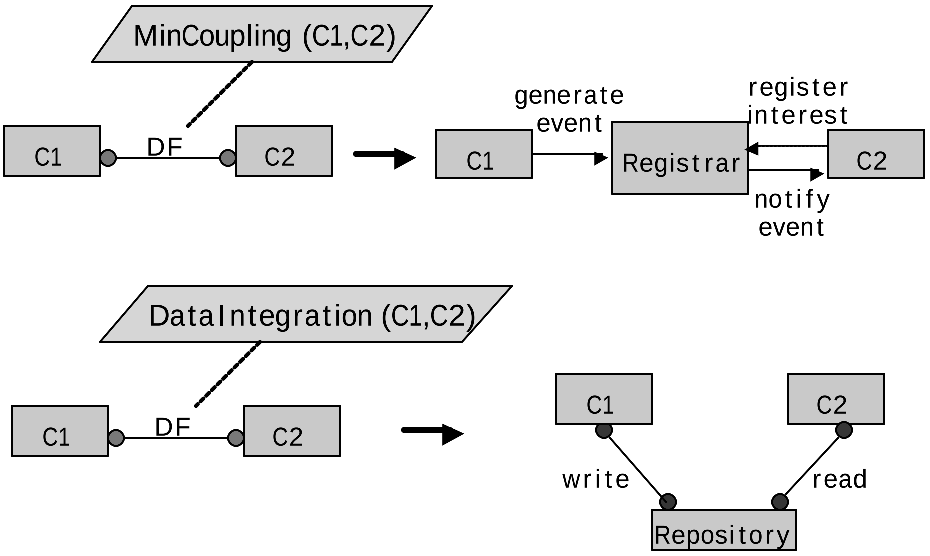
A few general patterns ... (4)



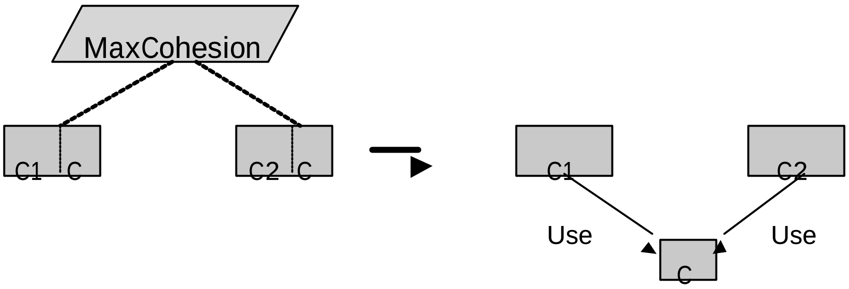
A few general patterns ... (5)

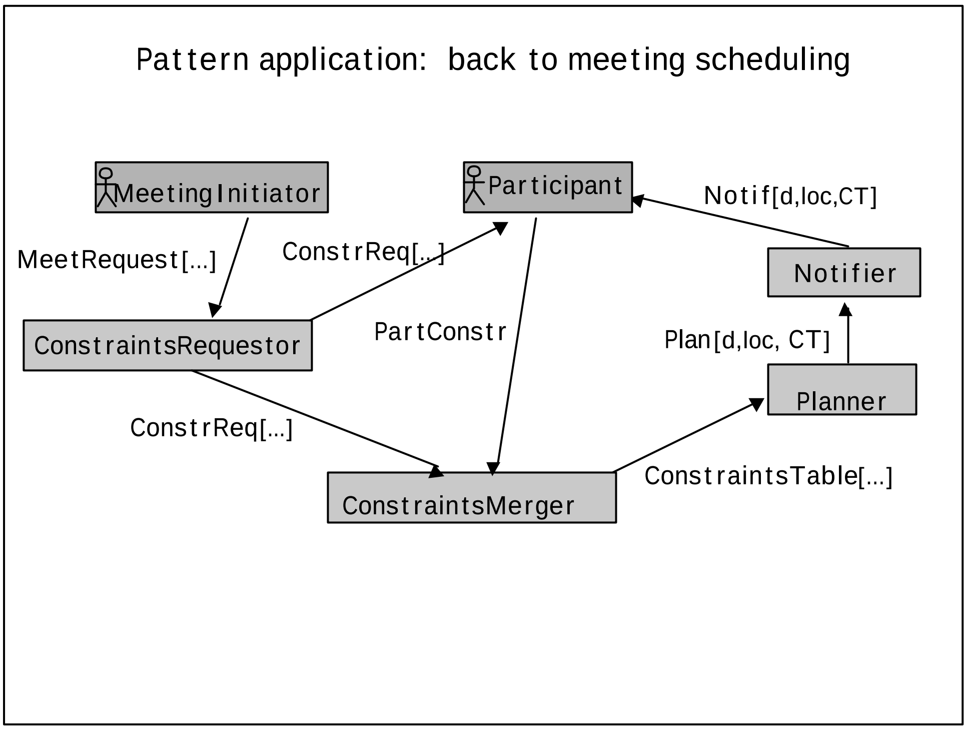


A few general patterns ... (6)



A few general patterns ... (7)





Pattern application: back to meeting scheduling

MeetingInitiator

Participant

Notif[d,loc,CT]

MeetRequest[...]

ConstrReq[...]

Notifier

ConstraintsRequestor

PartConstr

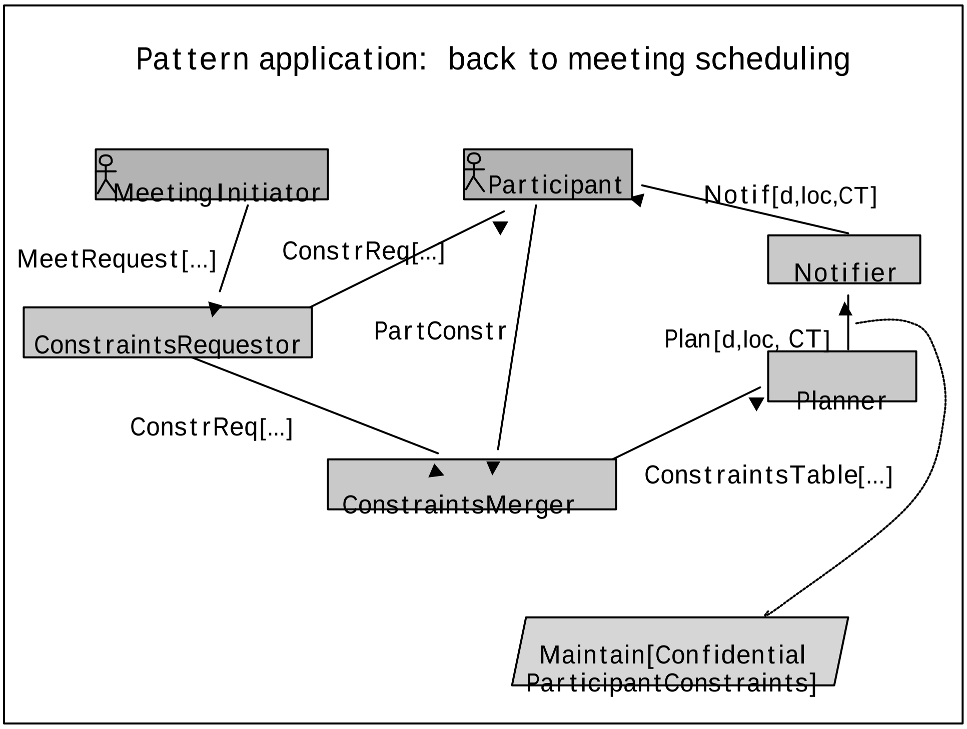
Plan[d,loc, CT]

Planner

ConstrReq[...]

ConstraintsTable[...]

ConstraintsMerger



Pattern application: back to meeting scheduling

MeetingInitiator

Participant

Notif[d,loc,CT]

MeetRequest[...]

ConstrReq[...]

Notifier

ConstraintsRequestor

PartConstr

Plan[d,loc, CT]

Planner

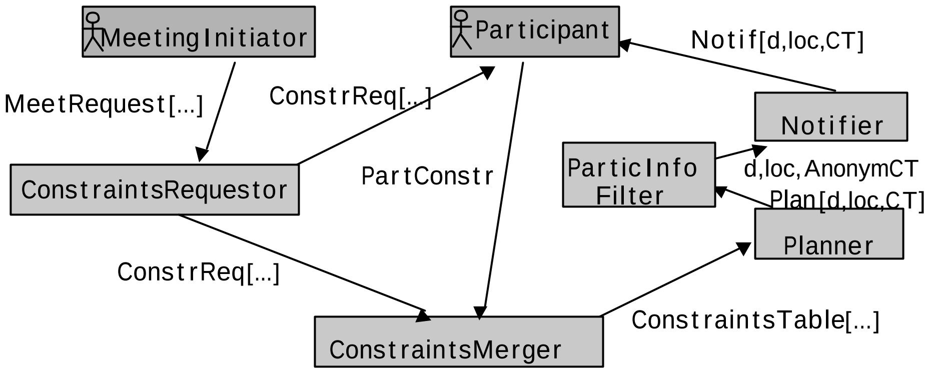
ConstrReq[...]

ConstraintsTable[...]

ConstraintsMerger

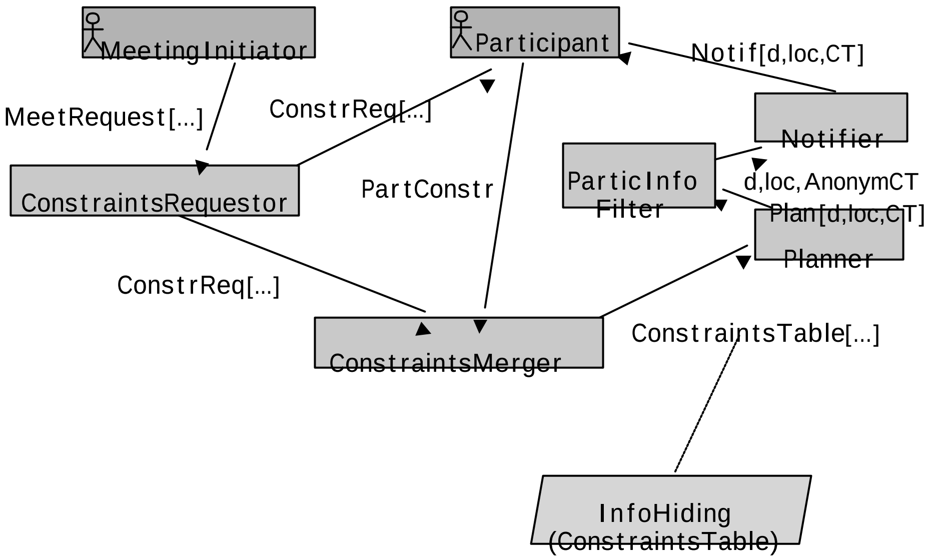
Maintain[Confidential
ParticipantConstraints]

Pattern application: back to meeting scheduling



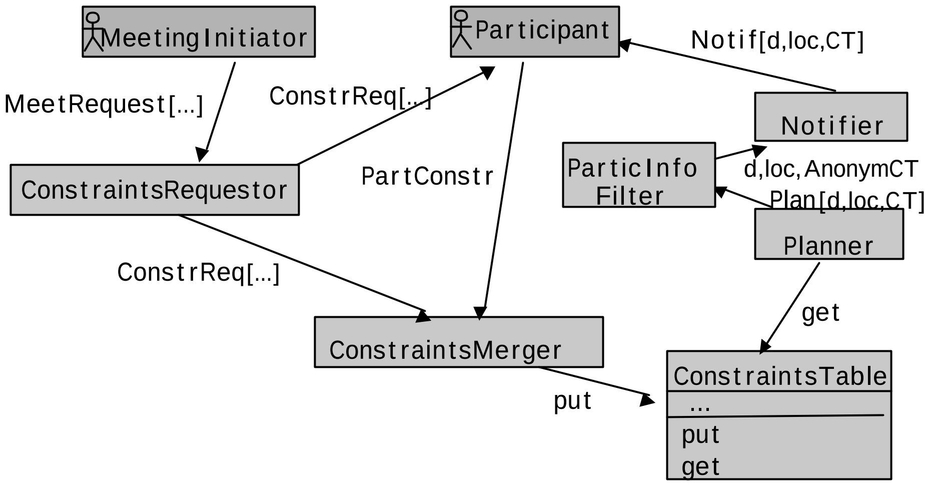
MeetingInitiator
Participant
Notif[d,loc,CT]
MeetRequest[...]
ConstrReq[...]
Notifier
ParticInfo Filter
d,loc, AnonymCT
ConstraintsRequestor
PartConstr
Plan[d,loc,CT]
Planner
ConstrReq[...]
ConstraintsTable[...]
ConstraintsMerger

Pattern application: back to meeting scheduling (2)



MeetingInitiator
Participant
Notif[d,loc,CT]
MeetRequest[...]
ConstrReq[...]
Notifier
ParticInfo Filter
d,loc, AnonymCT
ConstraintsRequestor
PartConstr
Plan[d,loc,CT]
Planner
ConstrReq[...]
ConstraintsTable[...]
ConstraintsMerger
InfoHiding (ConstraintsTable)

Pattern application: back to meeting scheduling (2)



MeetingInitiator

Participant

Notif[d,loc,CT]

MeetRequest[...]

ConstrReq[...]

Notifier

ConstraintsRequestor

PartConstr

ParticInfo Filter

d,loc,AnonymCT

Plan[d,loc,CT]

Planner

ConstrReq[...]

get

ConstraintsMerger

ConstraintsTable

...

put

get

put

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

ß

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

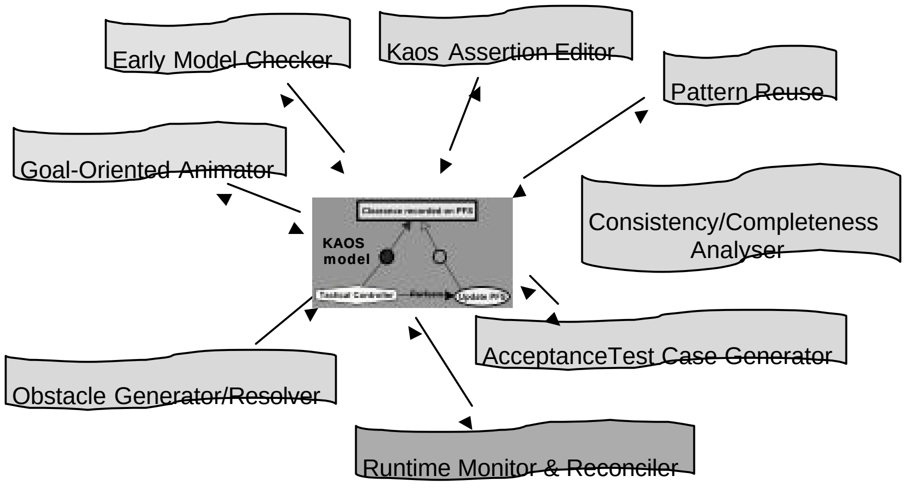
ß

view extraction through architectural net queries:

security view, accuracy view, reusability view, ...

Conclusion

Opportunities for goal-level tool support



Early Model Checker

Kaos Assertion Editor

Pattern Reuse

Goal-Oriented Animator

Consistency/Completeness Analyser

AcceptanceTest Case Generator

Obstacle Generator/Resolver

Runtime Monitor & Reconciler

Limitations & further work

- Only refinement-based:

  no bottom-up propagation of middleware requirements

  ß

  need for complementary abstraction patterns

- No derivation of interaction protocols

  ß

  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