Engineering Requirements for Mission-Critical Software Systems

Axel van Lamsweerde

University of Louvain
B-1348 Louvain-la-Neuve
avl@info.ucl.ac.be

Belgian Francqui Chair 2006-2007
Vrije Universiteit Brussel

Requirements on RE method ...

- Goal-oriented: to ensure that requirements meet system objectives -- including safety, security
- Incremental: for early analysis of partial models
- Constructive: for analyst guidance
- Model-based: for abstraction & structure
  - multiple models: for capturing multiple facets
- Mix declarative and operational styles as needed
- Formal when and where needed, but lightweight
Goal orientation...

- found in traditional methodologies for system engineering
  ("context analysis", "definition study", "participative analysis", ...)
- addressed by IEEE-Std-830
- ignored by UML, "but needed" say UMLers (Fowler, Cockburn)
- increasingly considered in RE research

KAOS: goal-oriented, model-driven RE

interviews  existing systems  documents

analysis  modeling

generation of RE deliverables

.html  .rtf  .pdf  .mif

© A. van Lamsweerde
Rich models require a multi-view approach

- Multiple views along the WHY, WHAT, WHO dimensions
  - intentional: functional & non-functional goals, goal diagrams
  - structural: conceptual objects, UML class diagrams
  - responsibility: system agents, context diagrams
  - functional: S2B services, operationalization diagrams and UML use cases
  - behavioral: system behaviors, scenarios & state machines (UML sequence and state diagrams)
- View integration: derivation links, consistency rules
  KAOS = Keep All Objectives Satisfied

What models?

- Goals
- Agents, responsibilities
- Objects
- Operations
What models? (2)

**Hazards**

**Threats**

**Interaction scenarios**

**Behaviors**

---

### Outline

- **Goal-oriented model building: an overview**
  - The goal, object, agent, and operation models
    - KAOS: a model building method in action
- **Formal reasoning about system models**
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
  - Conflict analysis
  - Goal-oriented model animation
  - Synthesizing behavior models from scenarios/goals
The goal model

- Intentional view of the system being modeled
- Goal = objective to be achieved by system ...
  - prescriptive statement of intent
  - "system": software + environment
    system-as-is, system-to-be
    “doors shall be closed while train is moving”
- ... unlike domain properties ...
  descriptive statements about environment
  “train is moving iff its physical speed is non-null”

Goals are specified in model annotations

Goal  Maintain [DoorsClosedWhileMoving]
Def  All train doors shall be kept closed at any time
     when the train is moving
[ FormalSpec  \forall \text{tr: Train}
    \text{Moving (tr) } \Rightarrow \text{tr.DoorsState = ‘closed’ } ]
[ Priority Highest ]
[ Source ... ]
[ Category ... ]

Optional formalization in real-time temporal logic
for formal reasoning (cf. below)
Goals are defined at different levels of abstraction

- Higher-level goals
  strategic, coarse-grained
  "50% increase of transportation capacity" (train control)
  "effective access to state of the art" (library system)

- Lower-level goals
  technical, fine-grained
  "acceleration command sent every 3 secs"
  "reminder issued by end of loan period if no return"

Goal categories

- Functional, quality, development goals
- Functional goals
  - used to build operational models about intended services: use cases, state machines, ...
    "passengers transported to their destination"
    "book request satisfied"
Goal categories: non-functional goals

- Quality goals
  - about quality of service: security, safety, accuracy, usability, performance, interoperability...
    "worst-case stopping distance maintained"
    "info about other borrowers kept confidential"
- Development goals
  - about quality of development: cost, deadline, variability, maintainability, reusability, etc.

Goal categories

- Goal categories: non-functional goals
  - Quality goals
    - about quality of service: security, safety, accuracy, usability, performance, interoperability...
      "worst-case stopping distance maintained"
      "info about other borrowers kept confidential"
  - Development goals
    - about quality of development: cost, deadline, variability, maintainability, reusability, etc.
Goal types

- **Behavioral goals**
  - can be established in a clear-cut sense
  - capture intended behaviors declaratively
  - goal satisfaction, formal reasoning
  - used for building operational models
    
    "Worst-case stopping distance maintained"

- **Softgoals**
  - cannot be established in clear-cut sense
  - capture preferred behaviors
  - goal satisficing, qualitative reasoning
  - used to compare alternative options
    
    "Reduce driver's stress & workload"

Behavior goals prescribe sets of desired behaviors

```
DoorsClosed
WhileMoving

moving closed -> stopped closed -> stopped open -> stopped closed -> moving closed
```

© A. van Lamsweerde
Behavioral goal: subtypes

- **Achieve / Cease goals**: generate behaviors
  
  \[ \text{CurrentCondition} \Rightarrow \text{eventually TargetCondition} \]

  *e.g.* **Achieve** [FastJourney]:
  
  If train at platform then **within 5 minutes** it is at next platform

- **Maintain / Avoid goals**: restrict behaviors
  
  always **GoodCondition**

  \[ \text{CurrentCondition} \Rightarrow \text{always GoodCondition} \]

  *unless NewSituation*

  *e.g.* **Maintain** [DoorsClosedWhileMoving]

  **never** **BadCondition**

  *e.g.* **Avoid** [TrainsOnSameBlock]

Use of goal types & categories

- **Lightweight specification patterns**

- **Heuristic rules for elicitation, validation, reuse, conflict management**, ...

  "Is there any conflict between Information goals and Confidentiality goals?"

  "Confidentiality goals are Avoid goals (on Knows predicates)"

  "Safety goals have highest priority in conflict resolution"

  more specific types & categories

  \[ \Rightarrow \text{more specific heuristics} \]
Goal achievement requires agent cooperation

- **Agent** = role (rather than individual) responsible for goal achievement
  - "safe transportation" ↔ on-board train controller, tracking system, station computer, passengers, train driver, ...
  - "book copy returned on shelves" ↔ borrower, staff, library software

- **Agent types:**
  - **software** (software-to-be, legacy software)
  - **devices**
  - **humans**

Goal achievement requires agent cooperation (2)

- Finer-grained goal ⇒ fewer agents required for goal achievement

- Requirement = goal assigned to single agent in software-to-be
  - "doorState = 'closed' while measured speed is non-zero" ↔ train controller
  - "acceleration command sent every 3 secs" ↔ station computer

- Expectation = goal assigned to single agent in environment
  - "passenger exits train when doors open at destination"
Goal-based satisfaction arguments

\[ R, E, D \vdash G \]

"in view of properties \( D \) of the domain, the requirements \( R \) will achieve goals \( G \) under expectations \( E \)"

\( R: \) doors get open at station

\( D: \) passenger can get out when doors are open

\( E: \) passenger gets out if doors open at destination

\( G: \) passenger gets out at destination station

Modeling goals: goal links

- Links relate goals to ...
  - other goals
    AND/OR refinement \( \Rightarrow \) goal contributions
    conflict \( \Rightarrow \) resolutions
  - other submodels \( \Rightarrow \) traceability
    reference \( \rightarrow \) objects
    responsibility \( \rightarrow \) agents
    operationalization \( \rightarrow \) operations
    coverage \( \rightarrow \) scenarios
    obstruction \( \rightarrow \) obstacles
Modeling goals: AND/OR refinement

- An AND-refinement of goal $G$ into subgoals $G_1, \ldots, G_n$ states that $G$ can be satisfied by satisfying $G_1, \ldots, G_n$
  - the set $\{G_1, \ldots, G_n\}$ is called refinement of $G$
  - the subgoal $G_i$ is said to contribute positively to $G$

- $\{G_1, \ldots, G_n\}$ is a complete AND-refinement of $G$ iff satisfying $G_1, \ldots, G_n$ is sufficient for satisfying $G$ in view of known domain properties
  \[
  \{G_1, \ldots, G_n, \text{Dom}\} \models G
  \]

- An OR-refinement of goal $G$ into refinements $R_1, \ldots, R_m$ states that $G$ can be satisfied by satisfying all subgoals in any of the alternative refinements $R_i$
  - the refinement $R_i$ is called alternative for $G$

The goal model shows contribution links and satisfaction arguments

```
<table>
<thead>
<tr>
<th>EffectivePassengersTransportation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RapidTransportation</td>
</tr>
<tr>
<td>SafeTransportation</td>
</tr>
<tr>
<td>And-refinement</td>
</tr>
<tr>
<td>FastJourney</td>
</tr>
<tr>
<td>HighFrequency</td>
</tr>
<tr>
<td>NoTrainCollision</td>
</tr>
<tr>
<td>DoorsClosed</td>
</tr>
<tr>
<td>BlockSpeedLimited</td>
</tr>
<tr>
<td>S2B</td>
</tr>
<tr>
<td>OR-refinement</td>
</tr>
<tr>
<td>FastRunWhen</td>
</tr>
<tr>
<td>SignalSetTo</td>
</tr>
<tr>
<td>WorstCaseStopping</td>
</tr>
<tr>
<td>NoTrainsOnSameBlock</td>
</tr>
<tr>
<td>Current</td>
</tr>
<tr>
<td>limited</td>
</tr>
</tbody>
</table>
```
Complete AND-refinement

- Getting complete refinements of behavioral goals is essential for requirements completeness
- Domain properties are often used for arguing about complete refinements
  - descriptive assertions attached to domain objects in the object model
  - classified as ...
    - domain invariants - known to hold in every state
      "train doors are either open or closed"
    - domain hypotheses - assumed to hold in specific states
      "railway tracks are in good conditions ..."

Domain properties in AND-refinements

```
DoorsClosedWhileMoving

DoorsClosedWhileNonZeroSpeed
NonZeroSpeed
Moving Iff
domain invariant
```
Building goal models: heuristics & tips

- Early discovery of goals ...
  - Analysis of current system
    ⇒ problems, deficiencies
    ⇒ goals of S2B: avoid / reduce / improve them
  - Search for intentional & prescriptive keywords in documents available, interview transcripts, etc.
    - in order to, so as to, so that, ...
    - has to, must, to be, must be, shall, ensure, want, motivate, expected to, ...
    - purpose, objective, aim, concern, ...
    refinement links: “in order to X the system has to Y“

Building goal models: heuristics & tips (2)

- Later discovery of goals ...
  - by abstraction (bottom-up):
    asking WHY? questions about...
    lower-level goals
    interaction scenarios being elicited
    other operational material available
  - by refinement (top-down):
    asking HOW? questions about goals available
  - by use of refinement patterns (cf. below)
  - by resolution of obstacles, conflicts (cf. below)
Building goal models: HOW and WHY questions

**HOW?**

- EffectivePassengersTransportation
  - RapidTransportation
  - SafeTransportation
    - FastJourney
      - FastRunWhenGoSignal
    - HighFrequency
    - NoTrainCollision
      - DistanceMaintained
      - WorstCaseStopping
    - DoorsClosedWhileMoving
      - SignalSetToGoPromptly
    - BlockSpeedLimited
      - Current

**WHY?**

Building goal models: heuristics & tips (3)

- Refine goals ... until when?
  ... until assignable to single agents as ...
    - requirement (software agent)
    - expectation (environment agent)

- Abstract goals ... until when?
  ... until boundary of system capabilities is reached
    e.g. EliminateGreenhouseEffect is beyond system’s capabilities
Goal refinement ... until when?

Maintain [WC-SafeDistanceBetwTrains]

Maintain [Safe Speed/AccelCom'ed]

Maintain [Safe TrainRespToComd]

...
Goal refinement ... until when?

Maintain[WC-SafeDistanceBetwTrains]

Maintain[Safe Speed/AccelCom’ed]  Maintain[Safe TrainRespToComd]

Mt[AccurateEstimate OfSpeed/Position]  Mt[SafeComdTo NextTrainFromEstim]

OnBoard TrainControl

Tracking System
Goal refinement ... until when?

Maintain[WC-SafeDistanceBetwTrains]

Maintain[Safe Speed/AccelCom’ed]  Maintain[Safe TrainRespToCmd]  ...  OnBoard TrainControl

Mt[AccurateEstimate OfSpeed/Position]  Mt[SafeCmdTo NextTrainFromEstim]

Tracking System

Goal refinement ... until when?

Maintain[WC-SafeDistanceBetwTrains]

Maintain[Safe Speed/AccelCom’ed] Maintain[Safe TrainRespToCom'd] ...

Mt[AccurateEstimate OfSpeed/Position] Mt[SafeComdTo NextTrainFromEstim]

Tracking System


Speed/Accel Control Communic Infrastruct
Building goal models: heuristics & tips (4)

- Do not confuse ...
  - goal ...
  - operation ...

Goal ≠ service from functional model (e.g. use case)
- services operationalize functional, leaf goals in refinement graph
- soft goals are often not operationalized in functional model but used to select among alternatives

Tip: past participle for goal name (state to be reached/maintained)
in infinitive for operation name (action to reach/maintain that state)

Building goal models: heuristics & tips (5)

- Do not confuse ...
  - OR-refinement ...
  - AND-refinement by case ...

cf. case analysis:
(Case1 or Case2) ⇒ X equiv (Case1 ⇒ X) and (Case2 ⇒ X)

OR-refinement introduces alternative systems to reach parent goal
AND-refinement by case introduces complementary, conjoined subgoals within same system
To avoid ambiguity in goal interpretation ...
- a precise & complete goal definition is essential
- grounded on shared system phenomena, and agreed upon by all stakeholders

**WHY goal models?**

- **Criterion for requirements completeness**
  
  set \( \text{REQ} \) of requirements is complete if for all \( G \):
  \[
  \{ \text{REQ}, \text{EXPECT}, \text{Dom} \} \models G
  \]

- **Criterion for requirements relevance**
  
  \( r \) in \( \text{REQ} \) is pertinent if for some \( G \):
  
  \( r \) is used in \( \{ \text{REQ}, \text{EXPECT}, \text{Dom} \} \models G \)

(Yue, 1987)
WHY goal models? (2)

- Drive the elaboration of requirements to support them
- Provide rich structuring mechanism:
  AND/OR refinement, abstraction
- Goal AND abstraction ⇒ requirements rationale
- Goal AND refinement ⇒
  - user-oriented structuring of requirements document
  - satisfaction arguments
  - rich traceability
    strategic objectives → technical requirements

WHY goal models? (3)

- Goal OR-refinement, OR-refinement
  ⇒ alternative options
- Support for evolution management
  higher-level goals ⇒ more stable concerns
  ⇒ multiple system versions within single model:
    common parent goals, different OR-branches
- Roots for conflict detection & resolution (cf. below)
- Roots for risk management (cf. below)
Fixing frequent misconceptions

- Goal-oriented ≠ top-down
- Goal-oriented ⇒ agent-oriented, scenario-oriented

the magic RE triangle (cf. end of these lectures)

What models?

- Goals
- Agents & responsibilities
- Objects
- Operations

on what?
Modeling conceptual objects

- Structural view of the system being modeled
- Object = set of instances of system-specific concept
  - can be distinctly identified & enumerated
  - share similar features (attributes, associations)
  - have specific behaviors from state to state
- Object specializations (at meta level):
  - entity: autonomous object
  - association: object dependent on objects it links
  - event: instantaneous object
  - agent: active object, controls behaviors

Conceptual objects & current instances

- Built-in semantic relation:
  InstanceOf (o, Ob) iff o is currently an instance of Ob
- Every concept must be defined by a Def annotation
  making it precise the conditions for an individual to
  satisfy InstanceOf (o, Ob)
  e.g. “A borrower is any person who has registered to the
      corresponding library for the corresponding period of time”
- Other object features:
  - attributes, associations ⇒ state of object instance
  - domain invariants = domain properties
  - initial state when InstanceOf (o, Ob) becomes true
Associations

- **Instance of Ass** = tuple of linked object instances, each playing corresponding role

- **Predicate notation**
  \[ \text{Ass}(o_1, ..., o_n) \text{ for } \text{InstanceOf}([o_1, ..., o_n], \text{Ass}) \]

\[ \begin{array}{ccc}
\text{Train} & \text{isOn} & \text{Block} \\
\text{On} & \text{holdsTrain} & \\
\text{InstanceOf} & \\
\text{On}(\text{tr}_2,\text{bl}_1) & \text{On}(\text{tr}_2,\text{bl}_1) & \\
\text{On}(\text{tr}_1,\text{bl}_3) & \\
\end{array} \]

Associations (2)

- **Multiplicities**
  from source instance, min/max number of target instances in current state

- **May encode some ...**
  domain properties
  "A train may be at one station at most at a time"
  requirements
  "A block may not accommodate more than one train at any time"

- **BUT ...**
  - mix prescriptive & descriptive assertions
  - most assertions are not expressible by multiplicities
    \[ \Rightarrow \text{need for domain invariants, goals/requirements} \]
Attributes

◆ Attribute Att of object Ob:
  - Function Att: Ob \( \rightarrow \) Range
  - Range is not a conceptual object we want to model
  - Elementary or structured ranges
  - attributes can be attached to entity, association, event, agent (like associations)

Built-in associations

◆ Specialization
  - Sub-class links
    InstanceOf (o, SubOb) ⇒ InstanceOf (o, SuperOb)
  - Multiple inheritance of attributes & associations on SuperOb

◆ Aggregation
  - Part-to-whole links
    InstanceOf (ag, Ag) ⇒ ag = Tuple (p1, ..., pn) with InstanceOf (pi, Parti)
  - Composition
    ag = Tuple (p1, ..., pn) and not InstanceOf (ag, Ag) ⇒ not InstanceOf (pi, Parti)
Modeling objects using UML

Modeling objects using UML (2)

© A. van Lamsweerde
Specifying conceptual objects through annotations

**Entity** Train

**Def** Any train circulating under control of the system-to-be.

**Has**
- CurrentLoc: PositionRange  % position of first car %
- DoorsState: {open, closed}  % attribute definition... %
- Speed: SpeedRange  % attribute definition... %
- Accel: AccelerationRange  % attribute definition... %
- Capacity [0..1]: # Natural  optional & rigid attribute

**DomInvar** Non-zero train acceleration implies non-zero speed. ...

**DomInit** A train appearing in the S2B has doors closed, zero speed and position XX.

domain properties

The object model is derivable from the goal model

Goal Maintain [BlockSpeedLimited]

InformalDef A Train should stay below the max speed the block can handle

FormalDef \( \forall \text{tr: Train, ts: TrackSegment} \quad \text{On (tr, ts)} \Rightarrow \text{tr.Speed} \leq \text{ts.SpeedLimit} \)

Systematic, no "hocus pocus" (as confessed by UML gurus)  \( \Rightarrow \) completeness & pertinence of object model
The object model is derivable from the goal model (2)

Goal: Maintain [WC-SafeDistanceBetwTrains]
   InformalDef: A Train should stay sufficiently far to avoid hitting the train in front in case of sudden stop
   FormalDef: ∀ tr1, tr2: Train
               Following (tr1, tr2) ⇒ tr1.Loc - tr2.Loc > tr1.WCS-Dist

Following

<table>
<thead>
<tr>
<th>Train</th>
<th>On</th>
<th>TrackSegment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed: SpeedUnit</td>
<td>Location</td>
<td>SpeedLimit: SpeedUnit</td>
</tr>
<tr>
<td>Loc: Location</td>
<td>WCS-Dist: Distance</td>
<td>...</td>
</tr>
</tbody>
</table>

Building object models: heuristics & tips

- For X: structural element in goal formulations...
  - X is defined in one single state
    ⇒ event
    InstanceOf(ev,Ev) denoted by Occurs(Ev)
  - X is active (controls pertinent behaviors)
    ⇒ agent
  - X is passive, autonomous (with distinguishable instances)
    ⇒ entity
  - X is passive, contingent upon other concepts (with distinguishable instances)
    ⇒ association
Building object models: heuristics & tips (2)

- For X: structural element in goal formulations...
  - Make X an attribute when...
    - instances of X are non-distinguishable
    - X is a function (yielding one single value when applied to some conceptual instance)
    - the range is not a concept you want to attach attributes/associations to
    - you don’t want to attach attributes/associations to X, specialize it, or aggregate/decompose it

![Diagram of object models]

Building object models: heuristics & tips (3)

- Should I attach an attribute X to an object in association or to the association?

![Diagram of object models]

to association if object instance can be unassociated: to avoid losing info e.g. who did borrow this copy
Building object models: heuristics & tips (4)

- For conceptual link \( X \) between "component" & "composite" objects, should \( X \) be... an aggregation? an association?
  - \( X \) has a domain-specific semantics
    \( \Rightarrow \) association
  - \( X \) has a domain-independent semantics
    \( \Rightarrow \) aggregation
  - Component & composite objects seem independent
    \( \Rightarrow \) association
  - Component object seems subordinate to composite
    \( \Rightarrow \) aggregation/composition

Building object models: heuristics & tips (5)

- Avoid frequent flaws in conceptual modeling ...
  - object attribute as "pointer" to another object

![Diagram](image-url)
Building object models: heuristics & tips (6)

- Avoid frequent flaws in conceptual modeling ...
  - monitoring/control information should be modeled in agent model (context diagram),
    not in structural model

Building object models: heuristics & tips (7)

- Avoid frequent flaws in conceptual modeling ...
  - dynamic information should be modeled in behavior model (scenarios, state machines),
    not in structural model
What models?

- **Goals**
- **Agents & responsibilities**
- **Objects**
- **Operations**

Modeling agents in KAOS

- Responsibility view of the system being modeled: who is doing what & why

- **Agent**:
  - Software, device, human
  - Role rather than individual
  - Active object, controls behaviors (operations)
  - Runs concurrently with others
  - Agent responsible for goal ⇒ must restrict system behaviors (Feather’87)
  - goal must be realizable by agent (Letier’02)
Goal realizability by an agent

A goal $G$ is realizable by agent $ag$ iff ...

(intuitively)
... given $ag$'s monitoring & control capabilities
it is possible for $ag$ alone to satisfy $G$
without more restrictions than required by $G$

(more formally)
... there exists a transition system $TS_{ag} = (Init, Next)$
on variables monitored & controlled by $ag$
such that $\text{RUN}(TS_{ag}) = \text{HISTORIES}(G)$

Causes of goal unrealizability

A goal $G$ is unrealizable by an agent when ...

• lack of monitorability of variables to be evaluated
• lack of controllability of variables to be constrained
• reference to future
• conditional unsatisfiability (weaker than inconsistency)
• unbounded achievement (liveness property)

(Letier & AvL, 2002)
Goal realizability by an agent: examples

Example 1: Realizable by TrainController
\[ \text{measuredSpeed} \neq 0 \Rightarrow \text{doorState} = \text{\textquoteleft}closed\textquoteright \]

Example 2: Not realizable by TrainController
\[ \text{Moving} \Rightarrow \text{DoorsClosed} \]

Modeling agents in KAOS (2)

- An agent is modeled & specified by its features ...
  - Name, Def
  - (Attributes/associations, DomInvar/Init \(\rightarrow\) object model)
  - Type: software or environment agent
  - Responsibility: links to goal model
  - Monitoring, Control: links to object model
    - state variables = attributes, associations from object model
  - Performs: links to operation model
  - Dependency links to other agents for goal achievement or successful operation performance [Yu'93]
Modeling agents in KAOS: example

Alternative agent assignments define alternative system boundaries

(Dardenne'93, Letier'02)
Load analysis from query on agent model

A useful, derivable diagram: context diagram

- Links agents through their interfaces
  - interface = monitored/controlled variables
    (attrib/assoc from object model)
  - link\((ag_1, ag_2)\) with label \(\text{var}\) generated from agent model iff
    \(\text{var}\) is controlled by \(ag_1\), monitored by \(ag_2\)
    \(\text{var}\) is monitored by \(ag_1\), controlled by \(ag_2\)

\(\text{variables monitored by } ag_1 \& \text{controlled by } ag_2\)

\(\text{variables controlled by } ag_1 \& \text{monitored by } ag_2\)

("problem diagrams" in Jackson 2001; Letier 2001)
An example of context diagram

```
Train/CurrentSpeed, Train/CurrentLoc
Train/Actuator
Train/ActuatedAcceleration

Tracking System
Train/MeasuredSpeed, Train/MeasuredLoc
Speed&Accel
Controller
Command/
CommandedAcceleration

OnBoard Controller
```

Deriving context diagrams from goals

- Many goals are of the form
  
  \[ G: \text{CurrentCondition [monitoredVariables]} \Rightarrow \text{eventually/always} \text{TargetCondition [controlledVariables]} \]

- Example: \( \text{tr.Speed} \neq 0 \Rightarrow \text{tr.DoorState = "closed"} \)
What models?

**Goals**

**Agents & responsibilities**

**Objects**

**Operations**

Modeling operations in KAOS

- Functional view of the system being modeled: what services are to be provided? (statics)

- Operation Op:
  - relation \( \text{Op} \subseteq \text{InputState} \times \text{OutputState} \)
  - Op must operationalize underlying goals (proof obligation, cf. below)
  - Op applications define state transitions (events)
  - concurrent with others
  - atomic: maps to state at next smallest time unit (operations with duration: use start/end events)
Specifying operations

- **Name, Def**
- **DomPre**: condition characterizing the class of input states in the domain
- **DomPost**: condition characterizing the class of output states in the domain
- **Links to other models**: Operationalization (goals), Input/Output (objects), Performance (agent)

```
OpenDoors \rightarrow WhileMoving
WhileMoving \rightarrow DoorsClosed

Operationalization (complete)
```

Specifying operationalizations

- An operationalization of G into Op is specified by:
  - **ReqPre**: necessary condition on Op's input states to ensure G (permission)
  - **ReqTrig**: sufficient condition on Op's input states to ensure G requires immediate application of Op provided DomPre holds (obligation)
  - **ReqPost**: condition on Op's output states to ensure G

- **Consistency rule**: ReqTrig ∧ DomPre ⇒ ReqPre
Specifying operations: example

Operation OpenDoors
  Def Operation controlling the opening of all train doors
  Input Train, Output Train/DoorsState

DomPre The train doors are closed
DomPost The train doors are open

ReqPre For DoorsClosedWhileNonzeroSpeed
  The train speed is 0

ReqPre For SafeEntry&Exit
  The train is at some station

ReqTrig For NoDelayToPassengers
  The train has just stopped and is at some station

Modeling operations: example
A useful, derivable diagram: use case diagram

- A use case outlines the operations an agent has to perform
  +: interactions with ...
  • the agents controlling operation inputs
  • the agents monitoring operation outputs
  +: optional (ill-defined) links ...
    to exceptions with preconditions ("extend")
    to sub-operations ("include")

⇒ A use case operationalizes the goals ensured by the operations in it

- Generation of use cases from the operation & agent models is straightforward

A derivable diagram: UML use case
A useful, derivable diagram: use case diagram

What models?

Goals

Agents & responsibilities

Objects

Operations
What models? (2)

Interaction scenarios

Behaviors

Modeling what could go wrong: obstacle analysis

- Problem: goals are often too ideal, will be violated (unexpected or malicious agent behaviors)
- Obstacle = condition on system for goal violation
  \[ \{O, \text{Dom}\} \models \neg G \quad \text{obstruction} \]
  \[ \text{Dom} \models \neg O \quad \text{domain consistency} \]
  \[ \exists \text{environment behavior } E \text{ s.t. } E \models O \quad \text{feasibility} \]
- Particular cases
  - obstruction of safetyGoals: obstacles = hazards
  - obstruction of securityGoals: obstacles = threats
Obstacle analysis for increased reliability & security

- Anticipate obstacles ...
  - deidealized goals, new goals
  - more complete, realistic requirements
  - more robust system


Goals provide anchors for hazard analysis

- Sector monitorable
- Flight plans known
- Aircraft positions known
- Controllers assigned to sectors
- Working communication means
- Communication pilot-controller possible
- Communication controllers adjacent sector possible
- Communication controllers same sector possible
Goals provide anchors for hazard analysis (2)

- **hazard as goal negation**
  - Sector not monitorable
  - Controllers not available
  - Aircraft positions not known
  - Communication problems
  - No communication with pilots
  - Communication problem between controllers

- **Flight manager failure**
- **Flight plans not known**
- **Wrong flight plans used**

- **hazard identification**
- **hazard resolution:** introduce new goals to avoid hazards

Obstacle models as goal-anchored fault trees

- **WorstCaseStoppingDistanceMaintained**
- **ReceivedCommand**
- **ExecutedByTrain**
- **SafeAcceleration**
- **Computed**
- **AccelerationSent**
- **InTimeToTrain**
- **SentCommand**
- **ReceivedByTrain**
Obstacle models as goal-anchored fault trees

WorstCaseStoppingDistanceMaintained

- SafeAcceleration
  - Computed
  - InTimeToTrain
  - ReceivedByTrain

- AccelerationCommand
  - NotSent
  - NotSentInTimeToTrain

- AccelerationCommand
  - NotReceived
  - NotReceivedInTimeByTrain

- Acceleration
  - NotSafe

- NotSent
- SentLate
- SentToWrongTrain
- NotReceived
- Corrupted
- ReceivedLate

Obstacle analysis: a real (but sad) example

Secor not monitorable

- Flight plans not known
  - Aircraft positions not known
  - No communication with pilots

- Flight manager failure
  - Wrong flight plans used

Controllers not available

Communication problems

- Communication problem between controllers
- Communication problem between controllers

ATC project @ CEDITI, completed March 2002
Uberlingen
mid-air collision, July 2002

Facts
- July 1st 2002, southern Germany
- DHL Boeing 757 x Russian Tu-154
- 71 people killed, incl. 52 children

Preliminary analysis shows:
- STCA out of order at Swiss ATC
- Only 1 controller on duty at crash time (the other one was taking a break) → controller overloaded
- Problem between air traffic handover between Switzerland and Germany for another flight landing
- German ATC failed to call Swiss ATC
- Conflict between Tu’s TCAS embedded system and tower’s order
- Pilot choice: Tower’s order prior to TCAS
- Discrepancies between screen displays and radar traces

Obstacle analysis: a real (but sad) example
Threat analysis requires extra-modeling: attacker's anti-goals & capabilities

ItemOrderedByBuyer \Rightarrow \Diamond_{\leq 7d} \text{ItemReceivedByBuyer}

ItemOrdered \Rightarrow \Diamond_{\leq 2d} \text{ItemPaid}

ItemPaid \Rightarrow \Diamond_{\leq 2d} \text{ItemSent}

ItemSent \Rightarrow \Diamond_{\leq 1d} \text{ItemReceived}

ItemPaid \Rightarrow \Diamond_{\leq 3d} \text{ItemSent}

\text{BELIEF}_S(\text{ItemPaid}) \Rightarrow \Diamond_{\leq 1d} \text{ItemSent}

\text{BELIEF}_S(\text{ItemPaid}) \Rightarrow \Diamond_{\leq 1d} \text{ItemSent}

\text{ItemReceived} \Rightarrow \Diamond_{\leq 1d} \text{ItemSent}

\text{BELIEF}_S(\text{ItemPaid}) \Rightarrow \Diamond_{\leq 1d} \text{ItemSent}

\text{PaymentReceived} \Rightarrow \Diamond_{\leq 1d} \text{NotificationSent}

\text{NotificationSent} \Rightarrow \Diamond_{\leq 1d} \text{NotificationReceived}

\text{NotificationReceived} \Rightarrow \Diamond_{\leq 1d} \text{NotificationReceived}

\text{NotificationReceived} \Rightarrow \Diamond_{\leq 1d} \text{NotificationReceived}

\text{NotificationReceived} \Rightarrow \Diamond_{\leq 16h} \text{FakeNotificationSent}
What models? (2)

Interaction scenarios
- Positive scenario: desired behavior (example)
- Negative scenario: undesired behavior (counterex)

Modeling interactions with scenarios

- Scenario Sc =
  - historical sequence of interaction events among agent instances
  - illustrates some way of achieving goal G: Sc is sub-history in set of behaviors prescribed by G
  - possibly composed of episodes (sub-scenarios)
  - interaction corresponds to application of operation by source agent, notified to target agent

- Positive scenario: desired behavior (example)
- Negative scenario: undesired behavior (counterex)
Scenarios as simple MSCs

Scenarios vs. goals: complementary benefits

- **Pros of scenarios:**
  - concrete
  - narrative
  - acceptance test data

- **Cons of scenarios:**
  - partial (cf. test coverage problem)
  - combinatorial explosion (cf. program traces)
  - procedural (unnecessary sequencing)
  - premature choice of system boundaries
  - requirements kept implicit

⇒ **Use of...**
  - scenarios for elicitation, validation (cf. below)
  - goals for declarative reasoning (cf. below)
Modeling behaviors: state machines

- State machine SM: \( \text{State} \times \text{Event} \rightarrow \text{State} \)
- State of an entity/association/agent instance:
  set of pairs (feature, value)
  \( \{ \text{att}_1 \rightarrow \text{val}_1, \ldots, \text{att}_n \rightarrow \text{val}_n, \text{assoc}_1 \rightarrow \text{link}_1, \ldots, \text{assoc}_p \rightarrow \text{link}_p \} \)
e.g.
  \( \{ \text{CurrSpeed} \rightarrow 0, \text{CurrLoc} \rightarrow X, \text{DoorsState} \rightarrow \text{'closed'}, \text{At} \rightarrow (\text{tr}, \text{st}) \} \)
- State of an entity/association/agent’s SM:
  set of states sharing same value for some behavioral attribute/association
e.g.
  \( \{ \text{CurrSpeed} \rightarrow 0, \text{CurrLoc} \rightarrow X, \text{DoorsState} \rightarrow \text{'closed'}, \text{At} \rightarrow (\text{tr}, \text{st}) \} \)
  \( \{ \text{CurrSpeed} \rightarrow 5, \text{CurrLoc} \rightarrow Y, \text{DoorsState} \rightarrow \text{'closed'}, \text{At} \rightarrow \text{nil} \} \)
belong to state "doorsClosed" of Train SM

Modeling behaviors: state machines (2)

- Guarded state transition:
  \( \text{St}_1 \xrightarrow{ev \text{ [guard] }} \text{St}_2 \)
  the object gets to state \( \text{St}_2 \) ...
  if it is in \( \text{St}_1 \) and \( ev \) occurs
  and only if the guard is true

\[
\begin{array}{c}
\text{DoorsOpening} \\
\text{[ AtStation and Speed } = 0 \text{ ]}
\end{array}
\]

\[
\begin{array}{c}
\text{doorsClosed} \\
\text{DoorsClosing}
\end{array}
\]

\[
\begin{array}{c}
\text{doorsOpen}
\end{array}
\]
**Modeling behaviors: state machines** (3)

- Nested states ...
  - sequential: super state is diagram composed of sequential sub-states (cf. Statecharts)

![State machine diagram]

- Concurrent: super state is diagram composed of concurrent sub-states (cf. Statecharts)

- Example: see next slide
Modeling behaviors with LTS

- **System behavior = composition of agent behaviors**
- **Each agent is modeled as a LTS**
  - Agents behave asynchronously but synchronize on shared events
  - Composition: by ||-operator
- **Pros & Cons**
  - Visual abstraction of behavior
  - Simple semantics
  - Executable
  - Rich opportunities for analysis, code generation
  - Hard to build & understand
Scenarios vs. behavior models

- A scenario defines paths in the behavior models
  - a path in each agent LTS
  - a path in the system's LTS (||)

Goals vs. scenarios

- A behavioral goal prescribes a set of scenarios
Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
- KAOS: a model building method in action
- Formal reasoning about system models
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
  - Conflict analysis
  - Goal-oriented model animation
  - Synthesizing behavior models from scenarios/goals

Model building in KAOS

1. Domain analysis: refine/abstract goals

SafeTransportation

NoTrainSameBlock
Model building in KAOS

1. Domain analysis: refine/abstract goals
   - SafeTransportation
   - NoTrainSameBlock

2. Domain analysis: derive/structure objects
   - Train on 0:1 Block

3. Domain analysis: concerns
   - DoorsClosedIFFnoZeroSpeed
   - BlockSpeedLimited

- Concerns
- Blocks
- NextBlock
- StationBlock
- Train
- MeasuredSpeed : Real
- doorsState : String
- Block
- BlockSpeed : Real
- TrainCars
- Following
- to
- from
Model building in KAOS

1. Domain analysis: refine/abstract goals
   - SafeTransportation
   - NoTrainSameBlock
   - SafeCord

2. Domain analysis: derive/structure objects
   - Train On Block

3. S2B analysis: enriched goals (alternatives)
Model building in KAOS

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
Model building in KAOS

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

- Obstacle & conflict analysis

Diagram:

- Train
- Block
- Command
- SafeTransportation
- SafeAcceler
- NoTrainSameBlock
- SafeComd
- Obstacle
- SignalNotVisible
- DriverUnresponsive
- BrakeSystemDown
- RegularResponsivenessCheck
- TrainStops IF StopSignal
- NoStopAtStopSignal
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

6. Operationalization & behavior analysis
Model building in KAOS

At any time: abstraction (e.g. from scenarios)
The goal-oriented RE method in action: another example

- "Water percolating into a mine has to be collected in a sump to be pumped out of the mine. Water level sensors shall detect when water is above a high and below a low level, respectively. A software pump controller shall switch the pump on when the water reaches the high water level and off when the water reaches the low water level. If, due to a failure of the pump, the water cannot be pumped out, the mine must be evacuated within one hour.

- The mine shall have other sensors to monitor the carbon monoxide, methane and airflow levels. An alarm shall be raised and the operator informed within one second when any of these levels reach a critical threshold so that the mine can be evacuated within one hour. In order to avoid the risk of explosion, the pump shall be operated only when the methane level is below a critical level.

- Human operators can also control the operation of the pump, but within limits. An operator can switch the pump on or off if the water is between the low and high water levels. A special operator, the supervisor, can switch the pump on or off without this restriction. In all cases, the methane level must be below its critical level if the pump is to be operated.

- Readings from all sensors, and a record of the operation of the pump, shall be logged for further analysis." (Joseph, 1996)
The goal-oriented RE method in action: another example (2)

- 1-3: First sketch of goal model (fragments)
  from intentional keywords: “to”, “in order to”, shall”, etc. in preliminary material, interviews, … (pre-canned here !)

The goal-oriented RE method in action: another example (3)

- 2-4: First sketch of object model (fragments)
The goal-oriented RE method in action: another example (4)

- First object model after Def of every preliminary goal:

```
Sump
WaterLevel: Depth
HasPump

Pump
Motor: {on, off}
Failure: Bool

Mine
MethaneLevel
CO-level
Airflow
Explosion

MethaneAlarm
AirflowAlarm
COAlarm

GasAlarm
... Raising

Operator
Informed

Miner
Inside
Operating

Operator
Informed

The goal-oriented RE method in action: another example (5)

- Modeling behavior of "interesting" domain objects
  e.g. Pump entity

```

```
NotOperating
StartUp
Stop
EndRepair
pumpProblem

Failure

Operating
NotPumping
pumpOn
pumpOff

Pumping
```

© A. van Lamsweerde
The goal-oriented RE method in action: another example (6)

- 1-3: Enrich first goal model
  asking WHY? & HOW? questions

```
NoPumpFailure
   NoPumpOnWhenNoWater
     PumpOnWhenLowWater
       NoOverflowedMine
         NoOverflowedSump
           SufficientPumpCapacity
             PumpOnWhenHighWater
               WaterPumpedWhenPumpOn
```

The goal-oriented RE method in action: another example (7)

- 1-3: Enrich first goal model
  asking HOW? questions: using refinement patterns

```
PumpOnWhenHighWater
   HighWaterDetected
     PumpOnWhenHighWaterDetected
       PumpSwitchOn
         WhenHighWaterDetected
           PumpOnWhenSwitchOn
```

milestone-driven refinement
resolve lack of controllability
The goal-oriented RE method in action: another example (8)

5: Elaborate agent model

WHO can take responsibility for WHAT?

The goal-oriented RE method in action: another example (9)

6: Operationalization of functional goals into use cases + operation specs
The goal-oriented RE method in action: another example (10)

- 1-5: Obstacle & conflict analysis:
  cfr. below:
  use of formal techniques for more accurate analysis

- 6: Scenario modeling
  for further elicitation (goal mining), validation, and
  behavior prescription for operations within use case

The goal-oriented RE method in action: another example (11)

- 6: Scenario modeling (cont’d)

- WaterLevel Sensor
  - WaterTooHigh
  - WaterOK

- PumpController
  - PumpOn
  - OK-On
  - pumpOff
  - OK-Off

- PumpActuator

© A. van Lamsweerde
The goal-oriented RE method in action: another example (12)

6: Inductive synthesis of state machines from scenarios: behavior of PumpController agent

Further reading on goal-oriented modeling


Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action

Formal reasoning about system models

- Checking goal refinements
- Deriving goal operationalizations
- Obstacle analysis
- Threat analysis
- Conflict analysis
- Goal-oriented model animation
- Synthesizing behavior models from scenarios/goals

Formal reasoning about system models ...

- To support more accurate analysis & derivations

- In KAOS:
  - optional "button": only when & where needed
  - declarative formalism for goals, requirements, assumptions, domain properties:
    - real-time temporal logic
  - more operational formalism for operations:
    - goal-oriented pre-/postconditions
Some bits of real-time linear temporal logic

- \( oP \): P shall hold in the **next** state
- \( \Diamond P \): P shall hold in **some future** state
- \( \Box P \): P shall hold in **every future** state
- \( P U N \): P shall hold in every future state **until** N holds
- \( P W N \): P shall hold in every future state **unless** N holds

Some bits of real-time linear temporal logic \((2)\)

Propositional connectives

\( \land, \lor, \neg, \rightarrow, \leftrightarrow \)

First-order language

quantifiers on object instance variables \( \forall, \exists \)

- \( P \Rightarrow Q : \Box (P \rightarrow Q) \)
- \( P \Leftrightarrow Q : \Box (P \leftrightarrow Q) \)
Some bits of real-time linear temporal logic \(^{(3)}\)

Real-time constructs:
\[\square_{\leq T} P: \quad P \text{ shall hold in every future state up to } T \text{ time units}\]
\[\Diamond_{\leq T} P: \quad P \text{ shall hold within } T \text{ time units}\]

Operators on past:
- \(\cdot P\): \(P\) did hold in the previous state
- \(\Diamond P, \oslash P\): \(P\) was always, since/\(\text{back}\) to \(O\)
- \(\oslash_{\leq T} P, \oslash_{\leq T} P, \text{etc}\)
- \(\Diamond P, \oslash P\)

\[\neg P \land P\]

Interpretation over historical state sequences

\(H\): historical sequence of states (behavior)
\(i\): time position (time is isomorphic to naturals)

\[(H, i) \models \Diamond P \iff (H, j) \models P \text{ for some } j \geq i\]

\[(H, i) \models \square P \iff (H, j) \not\models P \text{ for all } j \geq i\]
Interpretation over historical state sequences \( (2) \)

\[
\begin{align*}
(H, i) &\models P U N \iff (H, j) \models N \text{ for some } j \geq i \\
&\quad \text{and } (H, k) \models P \text{ for all } k: i \leq k < j \\
(H, i) &\models P W N \iff (H, i) \models P U N \text{ or } (H, i) \models \Box P \\
(H, i) &\models \Diamond_{\leq T} P \iff (H, j) \models P \text{ for some } j \geq i \\
&\quad \text{with dist } (i, j) \leq T
\end{align*}
\]

Goal specification patterns

**Achieve TargetCond**

\[
\text{CurrentCond } \Rightarrow \Diamond_{\leq T} \text{ TargetCond}
\]

**Cease TargetCond**

\[
\text{CurrentCond } \Rightarrow \Diamond_{\leq T} \neg \text{ TargetCond}
\]

**Maintain GoodCond**

\[
\Box \text{GoodCond} , \text{ CurrentCond } \Rightarrow \text{ GoodCond} \]

\[
\text{CurrentCond } \Rightarrow \text{ GoodCond} W \text{ NewCond}
\]

**Avoid BadCond**

\[
\Box \neg \text{BadCond} , \text{ CurrentCond } \Rightarrow \neg \text{ BadCond} \]

\[
\text{CurrentCond } \Rightarrow \neg \text{BadCond} W \text{ NewCond}
\]
Specifying goals in RT-LTL

Goal *Maintain* [DoorsClosedUntilNextStation]

FormalSpec $\forall \text{tr} \text{: Train, s : Station}$

$\text{At (tr, st)} \land o \neg \text{At (tr, st)} \Rightarrow$

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

Goal *Achieve* [FastJourneyBetweenStations]

FormalSpec $\forall \text{tr} \text{: Train, s : Station}$

$\text{At (tr, st)} \Rightarrow$

$\text{At (tr, next(st))}$

Goal-oriented spec of operations

Operation SendCommand

Input tr, tr’ : Train

Output cm: CommandMsg

$\neg \text{Sent (cm, tr)}$

$\text{Sent (cm, tr)}$

ReqPost for SafeCommandMsg

Following $(\text{tr}, \text{tr}') \rightarrow$

$\text{cm.Accel} \leq \text{F (tr, tr')} \land \text{cm.Speed} > \text{G (tr)}$

ReqTrig for CommandMsgSentInTime

$\Box_{\leq 0.5 \text{ sec}} \exists \text{cm}': \text{Sent (cm', tr)}$
Goals, objects, agents, operations: the semantic picture

- Multiple trigger conditions can get true in same state => true concurrency (no interleaving semantics)
- Pruning semantics: every behavior is allowed except those explicitly forbidden by the spec
- Frame axioms:
  - attrib/assoc variables not declared in Output declaration are left unchanged by the operation
  - for any op:
    \[ \text{DomPre}(\text{op}) \land \text{DomPost}(\text{op}) \Rightarrow \text{Performed}(\text{op}) \]
Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
- Formal reasoning about system models
  - Checking goal refinements
    - Deriving goal operationalizations
    - Obstacle analysis
    - Threat analysis
    - Conflict analysis
    - Goal-oriented model animation
    - Synthesizing behavior models from scenarios/goals

Correct goal refinement

- A set of goals \( \{G_1, ..., G_n\} \) correctly refines goal \( G \) in domain theory \( \text{Dom} \) iff

\[
\begin{align*}
\{G_1, ..., G_n, \text{Dom}\} & \models G & \text{completeness} \\
\{G_1, ..., G_n, \text{Dom}\} & \not\models \text{false} & \text{consistency} \\
\land_{j \neq i} G_j, \text{Dom} & \not\models G & \text{for each } i \in [1..n] & \text{minimality}
\end{align*}
\]
Checking goal refinements

- Getting correct AND-refinements of non-soft goals is essential e.g. for requirements completeness
- Aim: Show that goal refinement is correct
- Approach 1: Use TL theorem prover (STeP, ...)  
  heavyweight, non-constructive
- Approach 2: Use formal refinement patterns  
  to check, complete, explore refinements  
  lightweight, constructive (Darimont’96, Letier’02)

Using goal refinement patterns

- Build catalogue of refinement patterns that encode refinement tactics
- Prove patterns formally, once for all
- Reuse through instantiation, in matching situation
- Some frequent patterns:

  \[ C \rightarrow \diamond T \]

  \[ C \rightarrow \diamond M \quad M \rightarrow \diamond T \]

  milestone-driven

  \[ C \land D \rightarrow \diamond T \]

  case-driven / guard introduction
Checking goal refinements with patterns

Achieve [TrainProgress]
On (tr, b) ⇒ ♦ On (tr, next(b))

Achieve [ProgressWhenGo]
On (tr, b) ∧ Go[next(b)]
⇒ ♦ On (tr, next(b))

Achieve [SignalSetToGo]
On (tr, b) ⇒ ♦ Go[next(b)]

Achieve [TrainProgress]
On (tr, b) ⇒ ♦ On (tr, next(b))

missing subgoal !!
detectable automatically

guard introduction

Maintain [TrainWaiting]
On (tr, b) ⇒
On (tr, b) W On (tr, next(b))
Patterns provide guidance in formal refinement

\[ P \Rightarrow \Diamond Q \]

\[ P \land R \Rightarrow \Diamond Q \]

\[ \neg R \Rightarrow \Diamond R \]

\[ P \land \neg R \Rightarrow \Diamond R \]

from pattern catalogue

different designs

Patterns provide guidance in formal refinement (2)
Use formal pattern => reuse formal proof

1. \( P \Rightarrow \Diamond R \)
2. \( P \land R \Rightarrow \Diamond Q \)
3. \( P \Rightarrow P \land Q \)
4. \( P \Rightarrow (P \cup Q) \lor \Box P \)
5. \( P \Rightarrow \Diamond Q \lor \Box P \)
6. \( P \Rightarrow \Box R \lor (Q \land \Box P) \)
7. \( P \Rightarrow (\Diamond R \land \Diamond Q) \lor (\Diamond R \land \Box P) \)
8. \( P \Rightarrow (\Diamond R \land \Diamond Q) \lor (\Diamond R \land \Box P) \)
9. \( P \Rightarrow (\Diamond R \land \Diamond Q) \lor \Diamond \Diamond Q \)
10. \( P \Rightarrow (\Diamond R \land \Diamond Q) \lor \Diamond \Diamond Q \)
11. \( P \Rightarrow \Diamond Q \)

Patterns for generating refinements and assignments

- Goals need be refined until assignable to single agents
- Goals may be unrealizable by single agents ...
  - unmonitorable variable
  - uncontrollable variable
  - uncontrollable condition in environment
  - reference to future
  - unbounded Achieve goal
  \( \Rightarrow \) agent-driven tactics to resolve unrealizability
- Patterns yield alternative refinements + assignments
  (Letier & van Lamsweerde '02)
Generating refinements & assignments \(^{(2)}\)

Example: Introduce Accuracy goal

- **WHEN:**
  
  agent \(ag\) cannot monitor variable \(m\) to realize \(G[m]\)

- **WHAT:**
  - introduce monitorable image \(im\) of \(m\)
  - generate refinement...

\[
G \\
\uparrow \\
p(m) \Leftrightarrow q(im) \\
G[p(m)/q(im)]
\]

Generating refinements & assignments \(^{(3)}\)

Example: Introduce Accuracy goal

\[\text{MovingOnRunway} \Rightarrow \text{o ReverseThrustEnabled}\]

unmonitorable

by autopilot
Generating refinements & assignments (3)

Example: Introduce Accuracy goal

MovingOnRunway $\Rightarrow$ ReverseThrustEnabled

unmonitorable by autopilot

MovingOnRunway $\Leftrightarrow$ PlaneWeightSensed

PlaneWeightSensed $\Rightarrow$ ReverseThrustEnabled

Example:

Introduce Accuracy goal

Formal refinement patterns can be used informally

- Refinement by case
  - Applicable when goal achievement space can be partitioned into cases

GoalToBeEnsured

GoalToBeEnsured

GoalToBeEnsured

When Case 1

When Case 2

- Example of use:

ResourceRequestSatisfied

ResourceReserved

ResourceAllocated

When Available

When Not Available
Formal patterns can be used informally

- Refinement by milestone
  - Applicable when milestone states can be identified on the way to the goal’s target condition

- Example of use:

Informal use of patterns can reveal errors
Informal use of patterns can reveal errors

- Conflicts predicted
- Conflict
- The most direct route selected
- Sufficient route separation
- Air traffic monitored
- No unnecessary route changes
- Sector monitorable
- Sector monitored
- Situation always under control
- Predicted conflicts analyzed
- Unrealized Goal On Uncontrollable Condition
- Unrealized Goal On Unmonitorable Condition
- Predicted conflicts resolved
- Unrealized Goal On Unmonitorable Condition
- Unrealized Goal On Uncontrollable Condition

Milestone goals

Formal patterns can be used informally (3)

- Refinement towards goal realizability
  - Applicable when goal refers to quantities not monitorable/controllable by candidate agent

- UnrealizedGoalOnUnMonitorableCondition
- GoalOnMonitorableCondition
- UnrealizedGoalOnUnControllableCondition
- GoalOnControllableCondition

child node may be goal (incl. requirement, expectation) or domain property (invariant/hypothesis)
Formal patterns can be used informally \(^{(4)}\)

**Example of use:**

- **DoorsClosedWhileMoving**
- **Moving** \(\text{iff} \) **NonZeroSpeed**

- **NurseInterventionWhenCriticalPulseRate**
- **Alarm** \(\text{iff} \) **CriticalPulseRate**

Resolve lack of monitorability

Resolve lack of controllability

---

**Checking goal refinements ...**

- **Approach 3:** Roundtrip use of bounded SAT solver
  - Incremental check/debug of goal model fragments
  - On selected object instances (propositionalization)
  - With bounded traces (to be given)

  - Output:
    - **OK** (no counterexample found within trace bound)
    - **KO** + counter-example scenario satisfying
      \[ G_1 \land \ldots \land G_n \land \text{Dom} \land \neg G \]
      
    - Cf. tool demo
Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
- Formal reasoning about system models
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
  - Conflict analysis
  - Goal-oriented model animation
  - Synthesizing behavior models from scenarios/goals
Recall: TL Semantics of operations

\[
[| \text{op (arg, res)} |] =_{\text{def}} \text{DomPre (op)} \land \text{DomPost (op)}
\]

If \( R \in \text{ReqPre (op)} \) then

\[
[| R |] =_{\text{def}} (\forall^\ast) [| \text{op} |] \Rightarrow R
\]

If \( R \in \text{ReqTrig (op)} \) then

\[
[| R |] =_{\text{def}} (\forall^\ast) R \land \text{DomPre (op)} \Rightarrow [| \text{op} |]
\]

If \( R \in \text{ReqPost (op)} \) then

\[
[| R |] =_{\text{def}} (\forall^\ast) [| \text{op} |] \Rightarrow o R
\]

Correctness of goal operationalization

- A set of required conditions \( R_1, ..., R_n \) on operations correctly operationalizes a goal \( G \) iff

\[
[| R_1 |], ..., [| R_n |] = G \quad \text{completeness}
\]

\[
[| R_1 |], ..., [| R_n |] \neq \text{false} \quad \text{consistency}
\]

\[
G = [| R_1 |], ..., [| R_n |] \quad \text{minimality}
\]

- Every operationalization defines a proof obligation
Checking goal operationalizations

- **Aim:** Show that a goal operationalization is correct
- **Approach 1:** Use formal operationalization patterns to check, complete, explore operationalizations
  lightweight, constructive (Letier’02)
- **Approach 2:** Roundtrip use of bounded SAT solver with output
  - OK (no counterexample found within trace bound)
  - KO + counter-example scenario satisfying 
    \[
    [\vert R_1 \vert] \land ... \land [\vert R_n \vert] \land \text{Dom} \land \neg G
    \]

Formal operationalization patterns

- **Build a library of operationalization patterns** for frequent goal specification patterns
  - e.g. Achieve goals: \( C \Rightarrow \diamondsd T \quad C \Rightarrow \circ T \)
  - Maintain goals: \( C \Rightarrow T \quad C \Rightarrow \Box T \quad C \Rightarrow \text{TW N} \)
- **Prove pattern correctness once for all**
- **Reuse through instantiation, in matching situations**
Operationalization pattern: Mine Pump example

Goal Maintain [PumpSwitchOnWhenHighWaterDetected]

FormalDef

HighWaterSignal = 'On' ⇒ o PumpSwitch = 'On'

operational spec ?

---

Operationalization pattern: Mine Pump example

Goal Maintain [PumpSwitchOnWhenHighWaterDetected]

FormalDef

HighWaterSignal = 'On' ⇒ o PumpSwitch = 'On'

C ⇒ o T

C: HighWaterSignal = 'On'

T: PumpSwitch = 'On'

operational spec ?
 Operationalization pattern for $C \Rightarrow o T$

patterns proved correct once for all

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

Operation Op2
DomPre T
DomPost $\neg$ T
ReqPre for RootGoal $\neg C$

Operationalization pattern: Mine Pump example

$\text{HighWaterSignal} = 'On' \Rightarrow o \text{PumpSwitch} = 'On'$

C: $\text{HighWaterSignal} = 'On'$
T: $\text{PumpSwitch} = 'On'$
Operationalization pattern: Mine Pump example

\[
\text{HighWaterSignal} = 'On' \Rightarrow \lozenge \text{PumpSwitch} = 'On'
\]

\[C: \text{HighWaterSignal} = 'On'\]
\[T: \text{PumpSwitch} = 'On'\]

**Operation SwitchPumpOn**
- **DomPre**: PumpSwitch ≠ On
- **DomPost**: PumpSwitch = On
- **ReqTrig for RootGoal**: HighWaterSignal = 'On'

**Operation SwitchPumpOff**
- **DomPre**: PumpSwitch = On
- **DomPost**: PumpSwitch ≠ On
- **ReqPre for RootGoal**: HighWaterSignal ≠ 'On'

Operationalization catalogue for a taxonomy of goal patterns

- **Goal Patterns**
  - **Achieve**
    - **Unbounded Achieve**: \(C \Rightarrow \diamond T\)
    - **Bounded Achieve**: \(C \Rightarrow \diamond_{\infty} T\)
  - **Immediate Achieve**: \(C \Rightarrow O T\)
  - **State Invariance**
    - **Global Invariance**: \(\square (p \rightarrow q)\)
    - **"After" Invariance**: \(C \Rightarrow \square T\)
  - **Transition Invariance**
    - **"InBetween" Invariance**: \(C \Rightarrow T W N\)

**cf. Dwyer's spec patterns**

© A. van Lamsweerde
Operationalization catalogue for a taxonomy of goal patterns

Goal Patterns

**Achieve**
- Unbounded Achieve: $C \Rightarrow \diamond T$
- Bounded Achieve: $C \Rightarrow \diamond_{\infty} T$
- Immediate Achieve: $C \Rightarrow \bigcirc T$

**Maintain/Avoid**
- State Invariance: $C \Rightarrow \bigcirc T$
- Transition Invariance: $T \Rightarrow \bullet C$

Operationalization pattern for 'InBetween' goals

"T holds between $C$ and $N"$

$C \Rightarrow \bigcirc (T W (N \land T))$

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

Operation Op2
- DomPre: $T$
- DomPost: $\neg T$
- ReqPre for RootGoal: $\neg C B (N \land \neg C)$
pattern instantiation: Light Control System

\[ r.\text{FirstEntry} \Rightarrow \circ (r.\text{LightOn} \ W (r.\text{LastExit} \land r.\text{LightOn})) \]

\[ C: \ r.\text{FirstEntry} \]
\[ T: \ r.\text{LightOn} \]
\[ N: \ r.\text{LastExit} \]

Operation Op1
- \( \text{DomPre} \neg T \)
- \( \text{DomPost} T \)
- \( \text{ReqTrig for RootGoal} \neg C \)

Operation Op2
- \( \text{DomPre} T \)
- \( \text{DomPost} \neg T \)
- \( \text{ReqPre for RootGoal} \neg C \ B \ (N \land \neg C) \)

Pattern instantiation: Light Control System

\[ r.\text{FirstEntry} \Rightarrow \circ (r.\text{LightOn} \ W (r.\text{LastExit} \land r.\text{LightOn})) \]

\[ C: \ r.\text{FirstEntry} \]
\[ T: \ r.\text{LightOn} \]
\[ N: \ r.\text{LastExit} \]

Operation TurnLightOn
- \( \text{DomPre} \neg r.\text{LightOn} \)
- \( \text{DomPost} r.\text{LightOn} \)
- \( \text{ReqTrig for RootGoal} \ r.\text{FirstEntry} \)

Operation TurnLightOff
- \( \text{DomPre} r.\text{LightOn} \)
- \( \text{DomPost} \neg r.\text{LightOn} \)
- \( \text{ReqPre for RootGoal} \neg r.\text{FirstEntry} \ B \ (r.\text{LastExit} \land \neg r.\text{FirstEntry}) \)
Outline

- **Goal-oriented model building: an overview**
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
- **Formal reasoning about system models**
  - Checking goal refinements
  - Deriving goal operationalizations
- **Obstacle analysis**
  - Threat analysis
  - Conflict analysis
  - Goal-oriented model animation
  - Synthesizing behavior models from scenarios/goals

Goal obstruction by obstacles

- **Obstacle** = condition on system for goal violation

\[ \{ O, \text{Dom} \} \models \neg G \quad \text{obstruction} \]

\[ \text{Dom} \not\models \neg O \quad \text{domain consistency} \]

exists environment behavior \( E \) s.t. \( E \models O \) \( \quad \text{feasibility} \)

- **Domain-completeness of a set of obstacles \( O_1, \ldots, O_n \):**

\[ \{ \neg O_1, \ldots, \neg O_n, \text{Dom} \} \models G \]
Goal obstruction by obstacles (2)

- Correct refinement of obstacle $O$ in sub-obstacles $O_1, ..., O_n$:

  \[
  \{O_1, ..., O_n, \text{Dom}\} \models O \quad \text{completeness}
  \]

  \[
  \{O_1, ..., O_n, \text{Dom}\} \not\models \text{false} \quad \text{consistency}
  \]

  \[
  \{ \bigwedge_{j \neq i} O_j, \text{Dom} \} \not\models O \quad \text{for each } i \in [1..n] \quad \text{minimality}
  \]

Obstacle analysis

- Objective: Increase goal completeness through countermeasures to abnormal conditions

- For every leaf goal in refinement graph (requirement or expectation):
  - identify as many obstacles to it as possible
  - assess their likelihood & severity
  - resolve them according to likelihood/severity

(van Lamsweerde & Letier, 2000)
Obstacle identification

- For obstacle to goal $G$:
  - negate $G$;
  - find as many AND/OR refinements of $\neg G$ as possible in view of domain properties ...
  - ... until reaching obstruction preconditions that are feasible by the environment of the set of agents assigned to $G$

(goal-anchored fault-tree construction)

Obstacle identification: example

$$
\text{MotorReversed} \iff \text{MovingOnRunway}
\text{MovingOnRunway} \iff \text{WheelsTurning}
\text{MotorReversed} \iff \text{WheelsTurning}
$$
Obstacle identification: example

\[
\text{MotorReversed iff MovingOnRunway}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{obstruction}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{obstruction}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MotorReversed iff WheelsTurning}
\]

\[
\text{MovingOnRunway iff WheelsTurning}
\]
Obstacle identification: example

```
MotorReversed Iff MovingOnRunway

MotorReversed Iff WheelsTurning

MovingOnRunway Iff WheelsTurning

WheelsTurning Iff MotorReversed

MotorReversed Iff WheelsTurning

MovingOnRunway Iff WheelsTurning

WheelsTurning Iff MotorReversed

MotorReversed Iff WheelsTurning

OR-refinement (complete)

WheelsNotOut

WheelsBroken

Aquaplaning
```

Abductive generation of obstacles

- **Aim:** Find \( O \) such that
  \[
  0, \text{ Dom } \models \neg G \ , \ \text{ Dom } \not\models \neg O
  \]

- **Approach 1:** Use precondition calculus to get \( \neg G \) from Dom
  
  = regression of goal negation through domain theory

- **Approach 2:** Use formal obstruction patterns
  
  (van Lamsweerde & Letier, 2000)
Generating obstacles by regression

MovingOnRunway ⇒ o ReverseThrustEnabled

MovingOnRunway ⇔ WheelsTurning

WheelsTurning ⇒ o ReverseThrustEnabled

LTL: brief recall

\( oP \): 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

\( ◊P \): P shall hold in some future state

\( □_{≤T}P \): P shall hold in every future state up to T time units

\( ◊_{≤T}P \): P shall hold within T time units

+ past operators: "black" symbols

\( P ⇒ Q \) is \( □ (P ⇒ Q) \)

\( @ P \) \( \bullet (¬P) ∧ P \)
Generating obstacles by regression

Find precondition for obstruction of ...

\( \text{MovingOnRunway} \Rightarrow \text{WheelsTurning} \)

\[ \rightarrow \text{goal negation:} \]
\[ \diamond \text{MovingOnRunway} \land \neg \text{WheelsTurning} \]

\[ \rightarrow \text{regress through Dom:} \]
\[ ? \text{necessary conditions for wheels turning ?} \]
\[ \text{WheelsTurning} \Rightarrow \neg \text{Aquaplaning} \]

\[ \text{i.e.} \quad \text{Aquaplaning} \Rightarrow \neg \text{WheelsTurning} \]

\[ \rightarrow \text{RHS unifiable:} \]
\[ \diamond \text{MovingOnRunway} \land \text{Aquaplaning} \]

\[ \text{Warsaw obstacle} \]
Generating obstacles by regression

Find precondition for obstruction of ...

\[ \text{MovingOnRunway} \Rightarrow \text{WheelsTurning} \]

→ goal negation:

\[ \Diamond \text{MovingOnRunway} \land \neg \text{WheelsTurning} \]

→ regress through Dom:

? necessary conditions for wheels turning ?

\[ \text{WheelsTurning} \Rightarrow \neg \text{Aquaplaning} \]

i.e. \[ \text{Aquaplaning} \Rightarrow \neg \text{WheelsTurning} \]

→ RHS unifiable:

\[ \Diamond \text{MovingOnRunway} \land \text{Aquaplaning} \]

Warsaw obstacle
The regression procedure

- **Initial step:**
  - take \( O := \neg G \)
- **Inductive step:**
  - let \( A \Rightarrow C \) be the domain rule selected
    - with \( C \) matching some \( L \) in \( O \) whose occurrences are all positive in \( O \)
  - then \( \mu := \text{mgu}(L, C) \) (most general unifier)
  
\[
O := O[L / A.\mu]
\]

Every iteration produces finer sub-obstacles

... or use formal obstruction patterns

- **Very frequent pattern, used in this example:**

\[
\begin{array}{c}
\Diamond C \wedge \neg T \\
\Uparrow \\
\Diamond C \wedge \neg N \\
\end{array}
\]

obstacle

\[
\begin{array}{c}
T \Rightarrow N \\
\end{array}
\]

domain property:

necessary condition for target condition

- **Can be used to elicit domain properties as well**
Some frequent obstruction patterns

\[ C \Rightarrow \Box T \]

\[ \Diamond (C \land \Box \neg T) \]

\[ B \Rightarrow \Box \neg T \]

backward chain

\[ C \Rightarrow \Box \neg T \]

\[ \Diamond (C \land \Box \neg T) \]

\[ \neg \neg T U \neg P \]

starvation

\[ C \Rightarrow \Box T \]

\[ \Diamond (C \land \Box \neg T) \]

\[ \neg T W M \]

milestone

\[ C \Rightarrow \Box T \]

\[ \Diamond (C \land \Box \neg T) \]

\[ \neg T W M \]

milestone
Example of pattern instantiation

\[ \forall u: \text{User}, r: \text{Resource} \]
\[ \text{Requests} (u, r) \Rightarrow \Diamond \text{Gets} (u, r) \]

\[ \exists u: \text{User}, r: \text{Resource} \]
\[ \Diamond (\text{Requests} (u, r) \land \Box \neg \text{Gets} (u, r)) \]

\[ \exists u: \text{User}, r: \text{Resource} \]
\[ \Diamond (\text{Requests} (u, r) \land \Box (\neg \text{Gets} (u, r) \cup \text{coalition} (u, r))) \]

starvation

Obstacle assessment & resolution

- To assess likelihood & severity of identified obstacle: cfr. risk management techniques
- To resolve identified obstacle:
  - at RE time: model transformation
    - generate alternative resolutions
    - select "best" resolution based on ...
      - likelihood/severity of obstacle
      - other non-functional/quality goals
  - at run-time (for non-severe, occasional obstacles):
    obstacle monitoring, run-time resolution
    (Feather et al 1998, Robinson 2006, intrusion detection tools)
Generating obstacle resolutions

- Use of model transformation operators encoding resolution tactics
  - Goal substitution: consider alternative refinement of parent goal to avoid obstruction of child goal
    \[ \text{MotorReversed} \iff \text{WheelsTurning} \rightarrow \text{MotorReversed} \iff \text{PlaneWeightSensed} \]
  - Agent substitution: consider altern. responsibilities
    \[ \text{OnBoardTrainController} \rightarrow \text{VitalStationComputer} \]
  - Goal weakening
    \[ \text{TrafficControllerOnDutyOnSector} \rightarrow \text{TrafficControllerOnDutyOnSector or WarningToNextSector} \]

Generating obstacle resolutions (2)

- Model transformation operators (cont’d):
  - Goal restoration: enforce target condition at obstacle occurrence
    \[ \text{ResourceNotReturnedInTime} \rightarrow \text{ReminderSent} \]
    \[ \text{WheelsNotOut} \rightarrow \text{WheelsAlarmGenerated} \]
  - Obstacle prevention: new Avoid goal
    \[ \text{AccelerationCommandCorrupted} \rightarrow \text{Avoid [AccelerationCommandCorrupted]} \]
  - Obstacle mitigation: tolerate obstacle but mitigate its effects
    \[ \text{OutdatedSpeed/PositionEstimates} \rightarrow \text{Avoid [TrainCollisionWhenOutDatedTrainInfo]} \]
Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
- Formal reasoning about system models
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
    - Conflict analysis
    - Goal-oriented model animation
    - Synthesizing behavior models from scenarios/goals

Threat analysis requires an anti-model

- Anti-model = dual model, threats to model
- Anti-goal = malicious obstacle to satisfy attacker's objectives (and break security goals)
- Attacker has ...
  - capabilities: conditions she can monitor or control
  - system knowledge: goal model, domain properties

Software is now part of attacker's environment
Domain properties include software vulnerabilities

- Attack graph = refinement graph showing plan to achieve anti-goal in view of capabilities
Threat analysis

- Build attack graphs rooted on negation of security goals
  - Get initial anti-goals from application-specific security goals (e.g. instantiated patterns)
  - Identify attackers wishing them
  - Build refinement tree for these (regression, refinement patterns, plan generation)
- Derive new security goals as countermeasures to counter anti-goals in attack graph
  (van Lamsweerde, 2004)

Specification patterns for security goals

- Confidentiality goals
  Avoid [SensitiveInfoKnownByUnauthorizedAgent]
  \[ \forall ag: \text{Agent}, ob: \text{Object} \]
  \[ \neg \text{Authorized} (ag, ob.\text{Info}) \Rightarrow \neg \text{Knows} V ag (ob.\text{info}) \]
  \[ \text{Knows} V ag (v) \equiv \exists x: \text{Knows} ag (x = v) \]
  \[ \text{Knows} ag (P) \equiv \text{Belief} ag (P) \land P \]
  \[ \uparrow \]
  "P is in ag's memory"
- Other patterns for privacy, availability, integrity, authentication, non-repudiation, ...
  (De Landtsheer, 2005)
Application-specific instantiation of security goal patterns

Goal Avoid [SensitiveInfoKnownByUnauthorizedAgent]
\[ \forall ag: \text{Agent}, ob: \text{Object} \]
\[ \neg \text{Authorized}(ag, ob.\text{Info}) \Rightarrow \neg \text{KnowsV}_{ag}(ob.\text{info}) \]
↓
Web banking services
Object / Account [#, PIN]
Authorized (ag, acc) ≡
\[ \text{Owner}(ag, acc) \vee \text{Proxy}(ag, acc) \vee \text{Manager}(ag, acc) \]
↓
Goal Avoid [PaymentMediumKnownBy3rdParty]
\[ \forall p: \text{Person}, acc: \text{Account} \]
\[ \neg [\text{Owner}(p, acc) \vee \text{Proxy}(p, acc) \vee \text{Manager}(p, acc)] \]
\[ \Rightarrow \neg [\text{KnowsV}_p(acc.\text{Acc#}) \land \text{KnowsV}_p(acc.\text{PIN})] \]

© A. van Lamsweerde
### Application-specific instantiation of security goal patterns

**Goal Avoid [SensitiveInfoKnownByUnauthorizedAgent]**
\[ \forall ag: \text{Agent}, ob: \text{Object} \]
\[ \neg \text{Authorized}(ag, ob.\text{Info}) \Rightarrow \neg \text{KnowsV}_{ag}(ob.\text{info}) \]

↓  Web banking services

Object / Account [#, PIN]  sensitive object

Authorized (ag, acc) ⇔
\[ \text{Owner}(ag, acc) \lor \text{Proxy}(ag, acc) \lor \text{Manager}(ag, acc) \]

↓  Goal Avoid [PaymentMediumKnownBy3rdParty]

\[ \forall p: \text{Person}, acc: \text{Account} \]
\[ \neg [ \text{Owner}(p, acc) \lor \text{Proxy}(p, acc) \lor \text{Manager}(p, acc) ] \]
\[ \Rightarrow \neg [ \text{KnowsV}_{p}(acc.\text{Acc#}) \land \text{KnowsV}_{p}(acc.\text{PIN}) ] \]

### Get initial anti-goals

- Negate security goal instantiation to application-specific “sensitive” objects ...

**Goal Avoid [PaymentMediumKnownBy3rdParty]**

\[ \forall p: \text{Person}, acc: \text{Account} \]
\[ \neg \text{Authorized}(p, acc) \]
\[ \Rightarrow \neg [ \text{KnowsV}_{p}(acc.\text{Acc#}) \land \text{KnowsV}_{p}(acc.\text{PIN}) ] \]

↓  goal negation

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]

\[ \diamond \exists p: \text{Person}, acc: \text{Account} \]
\[ \neg \text{Authorized}(ag, acc) \land \text{KnowsV}_{p}(acc.\text{Acc#}) \land \text{KnowsV}_{p}(acc.\text{PIN}) \]
Get initial anti-goals

- Negate security goal instantiation to application-specific “sensitive” objects ...  

\[
\text{Goal Avoid } [\text{PaymentMediumKnownBy3rdParty}]
\forall p: \text{Person, acc: Account}
\neg \text{Authorized } (p, acc)
\Rightarrow \neg [\text{KnowsV}_p (\text{acc.Acc#}) \land \text{KnowsV}_p (\text{acc.PIN})]
\]

\[
\downarrow \quad \text{goal negation}
\]

\[
\text{Anti-Goal Achieve } [\text{PaymentMediumKnownBy3rdParty}]
\diamond \exists p: \text{Person, acc: Account}
\neg \text{Authorized } (p, acc)
\land \text{KnowsV}_p (\text{acc.Acc#}) \land \text{KnowsV}_p (\text{acc.PIN})
\]

Identify attackers wishing anti-goals

- For each initial anti-goal:
  - ask WHO might benefit from it
  - use of attacker taxonomies

\[
\text{Anti-Goal Achieve}[\text{PaymentMediumKnownBy3rdParty}]
\downarrow
\]

Insiders: Bank QA team
Organization-specific agents

Outsiders: Thieves
Hackers
Terrorists, ...
Build attack graph

For each (initial anti-goal, attacker):
build anti-goal refinement/abstraction graph ...

- Informally: by asking...
  WHY questions ⇒ parent anti-goals
  HOW questions ⇒ child anti-goals

Formally: by regression through ...

... domain properties
  \[ P \Rightarrow AG \]
  ⇒ anti-goal preconditions satisfiable in domain

... goal specs from attacked model
  ⇒ preconditions satisfiable by attacked software
Anti-goal refinement by regression through domain

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]
◊ ∃ p: Person, acc: Account
¬ Authorized (p, acc) ∧ KnowsV_p (Acc#) ∧ KnowsV_p (PIN)
↓ domain property as sufficient condition?
∀ p: Person, acc: Account
¬ Authorized (ag, acc) ∧ KnowsV_p (acc.PIN)
∧ (∃ x: Acc#) (Found (p, x) ∧ Matching (acc.PIN, x))
⇒ KnowsV_p (acc.Acc#) ∧ KnowsV_p (acc.PIN)
↓ anti-subgoal:
◊ ∃ p: Person, acc: Account
¬ Authorized (ag, acc) ∧ KnowsV_p (acc.PIN)
∧ (∃ x: Acc#) (Found (p, x) ∧ Matching (acc.PIN, x))
Anti-goal refinement by regression through domain

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]
◊ ∃ p: Person, acc: Account
¬ Authorized (p, acc) ∧ KnowsV_p (Acc#) ∧ KnowsV_p (PIN)
↓ dom prop as sufficient condition?
∀ p: Person, acc: Account
¬ Authorized (p, acc) ∧ KnowsV_p (acc.PIN)
∧ (∃ x: Acc#) (Found (p, x) ∧ Matching (acc.PIN, x))
⇒ KnowsV_p (acc.Acc#) ∧ KnowsV_p (acc.PIN)
↓ anti-subgoal:
◊ ∃ p: Person, acc: Account
¬ Authorized (p, acc) ∧ KnowsV_p (acc.PIN)
∧ (∃ x: Acc#) (Found (p, x) ∧ Matching (acc.PIN, x))

Build attack graph: refine until ...

◆ ... terminal conditions are reached ...
  - anti-requirements
    realizable in terms of attacker's capabilities
  - vulnerabilities of attackee
    properties of anti-domain
Refinement towards realizability by attacker: a known attack

PaymentMediumKnownBy3rdParty

PinKnown & MatchingAccountFound

AccountKnown & MatchingPinFound

PinKnown

MatchingAccount

Found

AccountKnown

MatchingPin

Found
Refinement towards realizability by attacker: a known attack

- PaymentMediumKnownBy3rdParty
- PinKnown & MatchingAccountFound
- AccountKnown & MatchingPinFound
- PinKnown
- MatchingAccountFound
- AccountKnown
- MatchingPinFound
- AccountCheckedForPinMatch
- CheckIteratedOnOtherAccountsIfNoMatch
- CheckRepeateable

Deriving countermeasures

- New security goals obtained by application of resolution operators, e.g.
  - Avoid [anti-goal]:
    - Avoid [AccountCheckRepeateableFromPin]
    - Avoid [PinCheckRepeateableFromAccount]
  - Make vulnerability condition unmonitorable by attacker
  - Make anti-req uncontrollable by attacker
Online shopping: functional goals

- ItemOrderedByBuyer $\Rightarrow \Diamond_{\leq 7d} \text{ItemReceivedByBuyer}$
- ItemOrdered $\Rightarrow \Diamond_{\leq 2d} \text{ItemPaid}$
- ItemPaid $\Rightarrow \Diamond_{\leq 2d} \text{ItemSent}$
- ItemSent $\Rightarrow \Diamond_{\leq 3d} \text{ItemReceived}$
- ItemPaid $\Rightarrow \Diamond_{\leq 8h} \text{PaymentReceived}$
- PaymentReceived $\Rightarrow \Diamond_{\leq 8h} \text{NotificationSent}$
- NotificationSent $\Rightarrow \Diamond_{\leq 8h} \text{NotificationReceived}$
- ItemReceived $\Rightarrow \Diamond_{\leq 1d} \text{ItemSent}$

Online shopping: a security goal

- ItemOrderedByBuyer $\Rightarrow \Diamond_{\leq 7d} \text{ItemReceivedByBuyer}$
- ItemOrdered $\Rightarrow \Diamond_{\leq 2d} \text{ItemPaid}$
- ItemPaid $\Rightarrow \Diamond_{\leq 2d} \text{ItemSent}$
- ItemSent $\Rightarrow \Diamond_{\leq 3d} \text{ItemReceived}$
- ItemPaid $\Rightarrow \Diamond_{\leq 8h} \text{PaymentReceived}$
- PaymentReceived $\Rightarrow \Diamond_{\leq 8h} \text{NotificationSent}$
- NotificationSent $\Rightarrow \Diamond_{\leq 8h} \text{NotificationReceived}$
- ItemReceived $\Rightarrow \Diamond_{\leq 1d} \text{ItemSent}$
Online shopping: anti-goal

ItemOrderedByBuyer ⇒ \( \Diamond_{\text{std}} \) ItemReceivedByBuyer

\( \Diamond_{\text{std}} \) ItemSent ⇒ ItemPaid

ItemOrdered ⇒ \( \Diamond_{\text{std}} \) ItemPaid

ItemPaid ⇒ \( \Diamond_{\text{std}} \) ItemSent

ItemSent ⇒ \( \Diamond_{\text{std}} \) ItemReceived

ItemPaid ⇒ \( \Diamond_{\text{std}} \) BELIEF_{S}(ItemPaid)

BELIEF_{S}(ItemPaid) ⇒ \( \Diamond_{\text{std}} \) ItemSent

ShippingCo

PaymentReceived ⇒ \( \Diamond_{\text{std}} \) NotificationSent

NotificationSent ⇒ \( \Diamond_{\text{std}} \) NotificationReceived

ItemPaid ⇒ \( \Diamond_{\text{std}} \) PaymentReceived

NotificationReceived ⇒ BELIEF_{S}(ItemPaid)

Seller

ItemOrdered

ItemOrderedByBuyer ⇒ \( \Diamond_{\text{std}} \) ItemReceivedByBuyer

\( \Diamond_{\text{std}} \) ItemSent ⇒ ItemPaid

ItemOrdered ⇒ \( \Diamond_{\text{std}} \) ItemPaid

ItemPaid ⇒ \( \Diamond_{\text{std}} \) ItemSent

ItemSent ⇒ \( \Diamond_{\text{std}} \) ItemReceived

ItemPaid ⇒ \( \Diamond_{\text{std}} \) BELIEF_{S}(ItemPaid)

BELIEF_{S}(ItemPaid) ⇒ \( \Diamond_{\text{std}} \) ItemSent

Seller

PaymentReceived ⇒ \( \Diamond_{\text{std}} \) NotificationSent

NotificationSent ⇒ \( \Diamond_{\text{std}} \) NotificationReceived

ItemPaid ⇒ \( \Diamond_{\text{std}} \) PaymentReceived

NotificationReceived ⇒ BELIEF_{S}(ItemPaid)

Seller
Online shopping: anti-goal model

- ItemOrderedByBuyer ⇒ \( \Box_{sd} \) ItemReceivedByBuyer
- ItemOrdered ⇒ \( \Box_{sd} \) ItemPaid
- ItemPaid ⇒ \( \Box_{sd} \) ItemSent
- ItemSent ⇒ \( \Box_{sd} \) ItemReceived
- ItemPaid ⇒ \( \Box_{sd} \) BELIEF_\( S \) (ItemPaid)
- BELIEF_\( S \) (ItemPaid) ⇒ \( \Box_{sd} \) ItemSent
- ItemPaid ⇒ \( \Box_{sd} \) NotificationReceived
- NotificationReceived ⇒ BELIEF_\( S \) (ItemPaid)
- PaymentReceived ⇒ \( \Box_{sh} \) NotificationSent
- NotificationSent ⇒ \( \Box_{sh} \) NotificationReceived
- ItemPaid ⇒ \( \Box_{sh} \) PaymentReceived
- Seller

Online shopping: anti-goal model

- ItemOrderedByBuyer ⇒ \( \Box_{sd} \) ItemReceivedByBuyer
- ItemOrdered ⇒ \( \Box_{sd} \) ItemPaid
- ItemPaid ⇒ \( \Box_{sd} \) ItemSent
- ItemSent ⇒ \( \Box_{sd} \) ItemReceived
- ItemPaid ⇒ \( \Box_{sd} \) BELIEF_\( S \) (ItemPaid)
- BELIEF_\( S \) (ItemPaid) ⇒ \( \Box_{sd} \) ItemSent
- ItemPaid ⇒ \( \Box_{sh} \) PaymentReceived
- NotificationReceived ⇒ BELIEF_\( S \) (ItemPaid)
- NotificationSent ⇒ \( \Box_{sh} \) NotificationReceived
- PaymentReceived ⇒ \( \Box_{sh} \) NotificationSent
- NotificationSent ⇒ \( \Box_{sh} \) NotificationReceived
- Attacker
- \( \Box_{sh} \) FakeNotificSent
Online shopping: new goal model with countermeasure

- ItemOrderedByBuyer ⇒ □ ItemReceivedByBuyer
- ItemOrdered ⇒ □ ItemPaid
- ItemPaid ⇒ □ ItemSent
- ItemSent ⇒ □ ItemReceived
- ... □ BELIEF(Seller, ItemPaid) ⇒ □ ItemSent
- PaymentConfirmed ∧ ConfirmRequested ⇒ □ BELIEF(Seller, ItemPaid)
- NotificationSent ⇔ □ NotificationReceived
- PaymentReceived ⇒ □ ConfirmRequested
- NotifReceived ⇒ □ ConfirmRequested
- Seller
- Paypal

Synthesizing attack graphs

- Builds proof showing realizability of anti-goal in view of attacker’s knowledge of environment & capabilities
- Capabilities = conditions that are monitorable and/or controllable
- Based on BDD representation of anti-goal
- Weakens powerful macro-agent by removal of capabilities, following BDD state-variable ordering
- Try proto tool at http://require.com

(Janssen & AvL, 2006)
Synthesizing attack graphs  (plan generation)

¬ ItemPaidByCustomer  ItemSentToCustomer

ShopKnowsItemPaidByCustomer

NotificationReceived

Attacker anti-goal:

¬ ItemPaidByCustomer ∧ ItemSentToCustomer

Attacker capabilities:

Controls  ItemPaidByCustomer, NotificationReceived
Monitors  --

Outline

◆ Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
◆ Formal reasoning about system models
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
  - Conflict analysis
    - Goal-oriented model animation
    - Synthesizing behavior models from scenarios/goals
Conflict analysis

- Divergence is most frequent case of conflicting goals, requirements or assumptions:
  potential logical inconsistency
- Goals $G_1, \ldots, G_n$ are divergent iff
  there exists a boundary condition $B$ :
  \[
  \{ B, \land_i G_i, \text{Dom} \} \models \text{false} \quad \text{inconsistency}
  \]
  \[
  \{ B, \land_{i\neq j} G_j, \text{Dom} \} \not\models \text{false} \quad \text{minimality}
  \]
  \[
  B \neq \neg \land_i G_i \quad \text{non-trivialness}
  \]
  exists system behavior $E$ s.t. $E \models B$ feasibility

Divergence: examples

- Regulations on California highways

Frequently involves security goals

Maintain[Reviewer Anonymity]:

Reviews ($r, \text{pap}, \text{rep}$) $\land$ AuthorOf ($a, \text{pap}$)
  $\Rightarrow \Box \neg \text{Knows (a, Reviews[r, pap, rep])}$

Achieve[Review Integrity]:

Reviews ($r, \text{pap}, \text{rep}$) $\land$ AuthorOf ($a, \text{pap}$)
  $\Rightarrow \Diamond \text{Gets (a, rep', pap, r)} \land \text{rep'} = \text{rep}$

Boundary condition: $\Diamond \exists r, \text{pap}, a, \text{rep}, \text{rep'}$

Reviews ($r, \text{pap}, \text{rep}$) $\land$ AuthorOf ($a, \text{pap}$)
  $\land \Diamond \text{Gets (a, rep', pap, r)} \land \text{rep'} = \text{rep}$
  $\land \text{French (r)} \land \neg \exists r' \neq r: \text{Expert (r')} \land \text{French (r')}$
Divergence: examples

- Regulations on California highways
- Frequently involves security goals

Maintain[Reviewer Anonymity]:
\[\text{Reviews} (r, \text{pap}, \text{rep}) \land \text{AuthorOf} (a, \text{pap}) \Rightarrow \Box \neg \text{Knows} (a, \text{Reviews}[r, \text{pap}, \text{rep}])\]

Achieve[Review Integrity]:
\[\text{Reviews} (r, \text{pap}, \text{rep}) \land \text{AuthorOf} (a, \text{pap}) \Rightarrow \Diamond \text{Gets} (a, \text{rep}', \text{pap}, r) \land \text{rep}' = \text{rep}\]

Boundary condition:
\[\Diamond \exists r, \text{pap}, a, \text{rep}, \text{rep}' \text{ Reviews} (r, \text{pap}, \text{rep}) \land \text{AuthorOf} (a, \text{pap}) \land \Diamond \text{Gets} (a, \text{rep}', \text{pap}, r) \land \text{rep}' = \text{rep} \land \text{French} (r) \land \neg \exists r' \neq r: \text{Expert} (r') \land \text{French} (r')\]
Conflict analysis \( (2) \)

- Detecting divergence:
  - by regression: derive \( B \) as precondition for \( \neg G_i \)
    from \( \{ \wedge_{i \neq j} G_j, \text{Dom} \} \)
  - by use of formal conflict patterns

- Resolving divergence: resolution operators
  - avoid boundary condition: \( \Box \neg B \)
  - restore divergent goals: \( B \Rightarrow \diamond \wedge_i G_i \)
  - anticipate conflict: \( P \Rightarrow \diamond \leq_T \neg P \)
  - weaken goals, specialize objects, etc.

\((\text{AvL et al, 1998})\)

Deriving boundary condition for conflict

By regression:

\( \text{AtStation} \wedge \circ \neg \text{AtStation} \Rightarrow \text{DoorsClosed} \ W \text{AtNext} \)

- (\( \text{Stopped} \wedge \text{Alarm} \)) \( \Rightarrow \text{DoorsOpen} \)

\( \Rightarrow \text{negate } G1: \)

\( \text{AtStation} \wedge \circ \neg \text{AtStation} \wedge \neg \text{AtNext} \ U (\text{DoorsOpen} \wedge \neg \text{AtNext}) \)

\( \Rightarrow \text{regress } \neg G1 \text{ through } G2: \)

\( \text{AtStation} \wedge \circ \neg \text{AtStation} \wedge \neg \text{AtNext} \ U (\bullet \text{Stopped} \wedge \bullet \text{Alarm} \wedge \neg \text{AtNext}) \)

boundary condition for conflict
Deriving boundary condition for conflict

By regression:
\[ \text{AtStation} \land \neg \text{AtStation} \Rightarrow \text{DoorsClosed} \quad W \text{AtNext} \]
\[ \bullet (\text{Stopped} \land \text{Alarm}) \Rightarrow \text{DoorsOpen} \]

\[ \Rightarrow \text{negate } G1: \]
\[ \text{AtStation} \land \neg \text{AtStation} \land \neg \text{AtNext} \ U (\text{DoorsOpen} \land \neg \text{AtNext}) \]

\[ \Rightarrow \text{regress } \neg G1 \text{ through } G2: \]
\[ \text{AtStation} \land \neg \text{AtStation} \land \neg \text{AtNext} \ U (\bullet \text{Stopped} \land \bullet \text{Alarm} \land \neg \text{AtNext}) \]
boundary condition for conflict
Deriving boundary condition for conflict

By regression:
\[ \text{AtStation} \land \neg \text{AtStation} \Rightarrow \text{DoorsClosed} \quad W \text{AtNext} \]
\[ \bullet (\text{Stopped} \land \text{Alarm}) \Rightarrow \text{DoorsOpen} \]
\[ \Rightarrow \text{negate } G1: \]
\[ \text{AtStation} \land \neg \text{AtStation} \land \neg \text{AtNext} \quad U (\text{DoorsOpen} \land \neg \text{AtNext}) \]
\[ \Rightarrow \text{regress } \neg G1 \text{ through } G2: \]
\[ \text{AtStation} \land \neg \text{AtStation} \land \neg \text{AtNext} \quad U (\bullet \text{Stopped} \land \bullet \text{Alarm} \land \neg \text{AtNext}) \]
boundary condition for conflict

Conflict analysis: a real example

ServeMorePassengers

\[ \text{Min[Distance Between Trains]} \]
\[ \text{Max[TrainSpeed]} \]
Conflict analysis: a real example

ServeMorePassengers

Min [Distance BetwTrains ]

Max [TrainSpeed]

Mt[CmdedSpeedClose ToPhysicalSpeed]

not too high

DistanceIncreased WithCmdedSpeed

SafeTransport

Conflict analysis: a real example

ServeMorePassengers

Min [Distance BetwTrains ]

Max [TrainSpeed]

Mt[CmdedSpeedClose ToPhysicalSpeed]

not too high

Mt[CmdedSpeedAbove 7mphOfPhysicalSpeed]

not too low

DistanceIncreased WithCmdedSpeed

SafeTransport

LimitedAccelerAbove 7mphOfPhysicalSpeed

SmoothMove
Conflict analysis: a real example

ServeMorePassengers
Min[Distance BetwTrains ]
Max[TrainSpeed]

Mt[CmdedSpeedClose ToPhysicalSpeed]
Mt[CmdedSpeedAbove 7mphOfPhysicalSpeed]

DistanceIncreased WithCmdedSpeed
SafeTransport

LimitedAccelerAbove 7mphOfPhysicalSpeed
SmoothMove

Conflict analysis: a real example

\( tr.CmAccel \geq 0 \Rightarrow \exists \ tr: tr.CmAccel \geq 0 \land f(DistObstacle) \leq 7 \)

Resolution:
\( tr.CmAccel \geq 0 \Rightarrow \exists \ tr: tr.CmAccel \geq 0 \land f(DistObstacle) \leq 7 \)

boundary condition for conflict

goal weakening
Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
- Formal reasoning about system models
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
  - Conflict analysis
- Goal-oriented model animation
  - Synthesizing behavior models from scenarios/goals

Model animation

- To check ...
  - model adequacy
  - model completeness
- By simulating model behavior on specific instances, in response to user's input events
- Ideally ...
  - in terms of domain visualizations
  - within some scope of concern (goal-oriented)
  - highlighting property violations (if any)
Animating partial vs. complete models

∀ tr:Train tr.moving ⇒ tr.doorState="closed"

RapidTransportation

SafeTransportation

FastJourney

HighFreq

RapidExit&Entrance

DoorsClosed

WhileMoving

BlockSpeedLimited

NoTrainCollision

FastRunWhenGoSignal

SignalSetToGo

WorstCaseStoppingDistanceMaintained

NoTrainOnSameBlock

GoToNextBlock

SetSignalGo

OpenDoors

StartTrain

…

Generating state machines from goal operationalizations

Goal model

Object model

Operation model

Compilation

FSMs class

Instantiation

FSMs instances

(Tran Van et al, 2004)
Step 1: Create SM structure from behavioral attr/associations

Step 2: Generate SM states
Step 3: Generate transitions & trigger events

Transition: openDoors(tr:Train)
sourceState: tr.doorState="closed"
targetState: tr.doorState="open"

Train

doorState
closed  open
moving
stopped moving ...

Step 4: Generate guard/trigger conditions

Transition: openDoors(tr:Train)
sourceState: tr.doorState="closed"
targetState: tr.doorState="open"
guard: ¬ tr.moving
trigCond: AtStation(tr) ∧ ¬ tr.moving

Train
doorState
closed  open
moving
stopped moving ...

© A. van Lamsweerde
Instantiating and initialising GSM machines

See demo of animator
Outline

- Goal-oriented model building: an overview
  - The goal, object, agent, and operation models
  - KAOS: a model building method in action
- Formal reasoning about system models
  - Checking goal refinements
  - Deriving goal operationalizations
  - Obstacle analysis
  - Threat analysis
  - Conflict analysis
  - Goal-oriented model animation

Synthesizing behavior models from scenarios/goals

Goals, scenarios, state machines: WinWin partners

- declarative
- many behaviors
- too abstract?

+ concrete examples
  - partial, few behaviors

+ executable
  - hard to build, understand
Scenarios as simple MSCs

- Positive Scenario 1:
  - Train Controller
  - Train Actuator Sensor
  - Passenger
  - Agent instance interaction event
  - Time
  - Partial order on events
  - Total order along timeline

- Positive Scenario 2:
  - Train Controller
  - Train Actuator Sensor
  - Passenger
  - Action: start
  - Event: alarm pressed
  - Action: emergency stop
  - Action: emergency open

- Negative Scenario 1:
  - Train Controller
  - Train Actuator Sensor
  - Passenger
  - Guard: open doors
  - Positive

Modeling behaviors with LTS

- An agent is modeled as a LTS
- System behavior = composition of agent behaviors
  - Agents behave asynchronously but synchronize on shared events
  - Composition: \( \parallel \)-operator

Composition operator \( \parallel \):

\[
P \parallel Q
\]
Scenarios vs. behavior models

- A scenario defines paths in the behavior models
  - a path in each agent LTS
  - a path in the system’s LTS

Goals, scenarios, state machines: WinWin partners

- Synthesizing behavior models from end-user scenarios
  - Brute force LTS induction from MSC+, MSC-
  - Generation of invariants as node decorations
- Optimizing the synthesis process
  - Heuristic search
  - Use of fluents & models of foreign components
  - **Goal injection**
- Goal mining from scenarios
  - WHY questions from MSC-
  - Automated inference of Maintain/Avoid goals
The LTS synthesis problem

- **Given a scenario collection** 
  \( S_c = (S_+, S_-) \) 
  showing typical examples of system usage

- **Synthesize the system as a composition of agent LTS** 
  \( S = \text{Alts}_1 \parallel ... \parallel \text{Alts}_n \) 
  such that it accepts \( S_+ \) and rejects \( S_- \).

Requirements on synthesis approach

- **For end-user involvement ...**
  - start from end-user +, - scenarios ... 
  - … and scenarios only ... 
    - avoid hMSC, state assertions, etc. 
  - ... but use such info when available

- **As scenario collection is incomplete ...**
  - support the elicitation of additional, “interesting” +, - scenarios 
  - be incremental: model refinable as further scenarios become available
Overview of synthesis approach

1. Generalization
   - interactive induction of system’s LTS
   - guided by generated scenario questions, classified as + or - by end-user
   - constrained by additional state info when available (state assertions, legacy components, goals, etc.)

2. System decomposition
   - standard automaton algorithms (e-moves, determinization, minimization)

(Damas et al, 2005)

LTS synthesis as grammar induction

- RPNI algorithm
  - learns a regular language
  - represented by a DFA
  - from +, - strings

- LTS synthesis problem
  - learn a behavior model
  - represented by system LTS
  - from +, - scenarios

Joint work with Pierre Dupont
Interactive RPNI induction: brute force approach

Input: A non-empty initial scenario collection \( S_c = (S_+, S_-) \)
Output: An automaton \( A \) consistent with an extended collection \( S_c = (S_+, S_-) \)

1. \( A \leftarrow \text{Initialize}(S_+) \)
2. \( (q, q') \leftarrow \text{ChooseStatePairs}(A) \)
3. \( A_{\text{new}} \leftarrow \text{Merge}(A, q, q') \)
4. \( \text{if Compatible}(A_{\text{new}}, S_-) \) then
   \( \text{ok} \leftarrow \text{true} \)
   \( Q \leftarrow \text{GenerateQuestion}(A, A_{\text{new}}) \)
   \( \text{if CheckWithEndUser}(Q) \) then
      \( S_+ \leftarrow S_+ \cup Q \)
   \( \text{else} \)
   \( S_- \leftarrow S_- \cup Q \)
   \( \text{ok} \leftarrow \text{false} \)
   \( \text{break} \)
5. \( \text{if ok then} \)
   \( A \leftarrow A_{\text{new}} \)

return \( A \)

Initial solution: the Prefix Tree Acceptor (PTA)

- Largest DFA accepting \( S_+ \), while rejecting \( S_- \).
- No generalization of observed behaviors.
- Red states are error states (from \( S_c- \)).
- Assumption: scenarios start in same state.

Prefix Tree Acceptor of the Train System
Generalize = merge state pairs

Consider all non-red states for pairwise merge
e.g. merging states 3 and 0:

Generalize: determinize

- Removes non-determinism (to learn a regular language)
- Produces further state merging
e.g. \{0,3\} has 2 outgoing “start” transitions
  => merge 7 and 2 for determinization
Generalize: determinize (2)

Example

\( \{2, 7\} \) has 2 outgoing “stop” transitions

=> merge 10 and 6 for determinization

Check generalization: generate scenario questions

Whenever new transition occurs from merged states
e.g. state 6 gets new outgoing “start” transition

prefix of scenario question = shortest history leading to 6
suffixes = gained continuations of 6 (only one here)
Generated scenarios

- Classified by end-user as positive or negative examples
- Constrain the generalization
- Elicit additional, “interesting” +/- scenarios
- Added to the initial scenario collection
- Theoretical characterization of learning sample for convergence

Checking intermediate solution

- Candidate state merge \((q, q')\) gets confirmed if
  - merging + determinization does not lead to merging error state & non-error state
  - all generated questions are classified as positive scenarios by user
- Otherwise, candidate solution \(A_{\text{new}}\) is discarded
- RPNI continues, choosing another pair \((q, q')\) for merge, until no more pair is available
Iterating on state pairs

\[
A \leftarrow \text{PTA}(S_+, S_-)
\]
for each state \( q \) with rank \( i \)
  
  for each state \( q' \) with rank \( j<i \)
    
    try
      
      \{
        \begin{align*}
          A_{\text{new}} & \leftarrow \text{merge} (A, q, q') + \text{determinization} \\
          \text{generate scenario questions}
        \end{align*}
      \}
      
      A \leftarrow A_{\text{new}}
    
    catch (IncompatibilityException) { // keep A }

Goals, scenarios, state machines: WinWin partners

- Synthesizing behavior models from end-user scenarios
  - Brute force LTS induction from MSC+, MSC-
  - Generation of invariants as node decorations
- Optimizing the synthesis process
  - Heuristic search
  - Use of fluents & models of foreign components
  - Goal injection
- Goal mining from scenarios
  - WHY questions from MSC-
  - Automated inference of Maintain/Avoid goals
Fluents: a bridge between declarative and operational worlds

**Fluents - from the Event Calculus:**

“Fluents - time varying properties of the world - are true at particular time-points if they have been initiated by an action occurrence at some earlier time-point and not terminated by another action occurrence in the meantime.”

*(Miller & Shanahan, 1999)*

Fluents: a bridge between declarative and operational worlds  

Fluent \( F = \langle I_{Fl}, T_{Fl} \rangle \)

- initially \( \text{InVal}_{Fl} \)

- \( I_{Fl} \): set of initiating events
- \( T_{Fl} \): set of terminating events
- \( \text{InVal}_{Fl} \): initial value of \( F \)

**Example**

- fluent moving = \( \langle \{\text{start}\}, \{\text{stop}, \text{emergency stop}\} \rangle \)
- initially **false**

- fluent doorsClosed = \( \langle \{\text{close doors}\}, \{\text{open doors, emergency open}\} \rangle \)
- initially **true**
Fluent value at node on single LTS path

Fluent Fl is **true** after finite LTS execution E ending in state q iff
- E is empty and Fl holds initially
- the last event of E belongs to the initiating events
- the last event ev of E does not belong to the terminating events and the fluent is true after $E'$, where $E = E'$

Fluent value at node on multiple paths

Not necessarily either true or false

emergency = < {alarm propagated}, {start} > initially **false**
- value at state 0 after empty execution is false
- value at state 0 after execution <a. propagated, e.open, close> is true
=> value of emergency at state 0 will be set to top
Decoration algorithm: spec

- Given
  LTS model \((Q, \Sigma, \delta, q_0)\)
  fluent \(Fl = \langle \text{Init}_F, \text{Term}_F \rangle\) initially \(\text{Init}_F\)

- Find
  function \(\text{decor}: Q \rightarrow \text{bool}_{abs}\) defined by
  \[
  \text{decor}(q) =
  \begin{cases}
    \text{true} & \text{iff } Fl \text{ is true for all LTS executions reaching state } q \\
    \text{false} & \text{iff } Fl \text{ is false for all LTS executions reaching state } q \\
    \text{top} & \text{iff } Fl \text{ is true for some LTS executions reaching state } q \text{ and false for others} \\
    \text{bottom} & \text{iff state } q \text{ is not reachable from the initial state}
  \end{cases}
  \]

Decoration algorithm: general idea

- Apply propagation rule for each transition until fixpoint is reached

  \[
  \text{if } \text{decor(source)} \neq \text{bottom} \text{ then}
  \begin{cases}
    \text{if } \text{event } \in \text{Init}_F \text{ then}
      \text{newVal } \leftarrow \text{true} \\
    \text{else}
      \text{if } \text{event } \in \text{Term}_F \text{ then}
        \text{newVal } \leftarrow \text{false} \\
      \text{else}
        \text{newVal } \leftarrow \text{decor(source)}
  \end{cases}
  \text{decor(target) } \leftarrow \text{sup} \text{decor(target), newVal}
  \]

- Keeps track of set toExplore of states that have been updated & where propagation rule should be applied
Initial solution

fluent moving = <{start}, {stop, emergency stop}> initially false

toExplore = { } 
toExplore = {0}

toExplore = {1,3,4}

Step 1 - Selecting state 0

fluent moving = <{start}, {stop, emergency stop}> initially false
Step 2 - Selecting state 1

toExplore = \{1,3,4\}

fluent moving = \{\text{start}, \{\text{stop, emergency stop}\}\} \text{ initially false}

Next steps

- Continue while toExplore is not empty
  - no further change because fixpoint is reached
- Can be extended to handle multiple fluents at once
- Example
  fluent moving = \{\text{start}, \{\text{stop, emergency stop}\}\} \text{ initially false}
  fluent doorsClosed = \{\text{close doors}, \{\text{open doors, emergency open}\}\} \text{ initially true}
Goals, scenarios, state machines: WinWin partners

- Synthesizing behavior models from end-user scenarios
  - Brute force LTS induction from MSC+, MSC-
  - Generation of invariants as node decorations

Optimizing the synthesis process
- Heuristic search
- Use of fluents & models of foreign components
- Goal injection

Goal mining from scenarios
- WHY questions from MSC-
- Automated inference of Maintain/Avoid goals

Optimizing the synthesis process – Why?

- Inefficiency of exhaustive generate-and-test
- Risk of overgeneralization
  - without questions (if no user available)
  - role of negative scenarios, non intuitive in first place
- Too many questions (often for scenarios to be rejected)
  - On train example: 20 questions (3 to be accepted and 17 to be rejected)
- Additional state information
  - we don’t want to make it mandatory …
  - … but we want to use it when available
Optim 1: Heuristic generalization

- For an intermediate solution A
  - Kernel states were shown to be incompatible with each other
  - Fringe states are located at one transition of a kernel state

- Merge fringe state with kernel one, in some order
  - questions here: (4,0); (4,1); (4,2) rejected; (5,0) accepted

- Select pairs (fringe, kernel) according to heuristic function based on shared continuations
  - questions here: (5,2) rejected, (5,0) accepted
  - 3 questions only on the entire train example (2 accepted and 1 rejected)

Optim 2a: Constrain induction with fluents

- Decorate PTA using fluent definitions
  - fluent moving = «{start}, {stop, emergency stop}» initially false

- Constrain induction: avoid merging states with inconsistent decorations
  - e.g. avoid merging state where train is moving with state where train is not moving
Optim 2b: Constrain induction with models of foreign components

Constrain induction: avoid merging states with different foreign component states
e.g. Doors are either opened or closed

Generalizing Optimization 2 to constrain induction

- Avoid merging...
  - error states AND non-error states
  - inconsistent decorations
  - incompatible legacy states
  - ...
  ↓
- Define equivalence relation on PTA states
- Avoid merging states NOT in same equivalence class
Optimization 3: Constrain induction with goals

- Back to induction …
  - Importance of negative scenarios in avoiding overgeneralization
  - Many negative scenarios ⇒ few generated questions
- Goals …
  - capture sets of scenarios
  - Avoid/Maintain goals capture many negative scenarios

Maintain/Avoid goals

- Technique available to model check LTS against safety properties [Giannakopoulou & Magee, 2003]
  builds a tester from safety property

e.g. DoorsClosedWhileMoving:
  ? (moving ? doorsClosed)
  negative scenario = path leading to error state
**Using Maintain/Avoid goals to constrain induction**

Goal tester is used to “lock” PTA against goal
- drastically reduces number of generated questions
- synthesized system LTS is ensured to never violate the goal
- use of NASA’s FLTLtoBüchi translator

[Damas et al, ACM FSE’2006]

---

**Goals, scenarios, state machines: WinWin partners**

- Synthesizing behavior models from end-user scenarios
  - LTS induction from MSC+, MSC-
  - Interaction through scenario questions only
  - Generation of invariants as node decorations
- Optimizing the synthesis process
  - search strategy
  - use of legacy components (if any)
  - goal injection
- Goal mining from scenarios
  - WHY questions from negative scenarios
  - Automatic inference of Maintain/Avoid goals
Eliciting goals from negative scenarios

- User can explain WHY scenario should be rejected
  - Make underlying goal explicit
  - Later, no question contradicting goal will be asked
  - Derived LTS will meet the goals

Goal explaining WHY NOT: ? (Moving → doorsClosed)

Goal mining from scenarios

- Inductive learning from examples
  - take scenarios as + / - examples
  - represent them as MSCs, clean them
  - generate candidate assertions...
    - covering all + examples
    - excluding all - examples
  STRIPS-like generation + generalization rules
  EDT = models of assertions generated

(van Lamsweerde & Willemet'98)
Goal mining from scenarios (2)

:Controller

arrival

doorsOpening

doorsClosing

start

arrival

doorsOpening

:Train

Goal mining from scenarios (3)

:Controller

¬ At

At, Closed

doorsOpening

At, Open

doorsClosing

At, Closed

start

¬ At, Closed

arrival

At, Closed

doorsOpening

At, Open
Goal mining from scenarios (4)

:Controller

¬ At
At, Closed
At, Open
At, Closed
¬ At, Closed
At, Closed
At, Open

:Train

arrival
doorsOpening
doorClosing
start
arrival

At \land o \neg At 
\Rightarrow Closed \land At

Goal mining from scenarios (5)

:Controller :Train :Passenger

arrival doorsOpening
doorClosing move
arrival doorsOpening

At (tr, st) \land o \neg At (tr, st) \Rightarrow 
tr.Doors = "closed" \land W At (tr, next(st))
Another, fluent based technique for inferring Maintain/Avoid Goals

- **Input**
  - set of positive scenarios
  - definition of fluents

- **Output**
  - for each agent, a set of goals \( G = \Box A \)
    
    \( A: \) disjunction of fluents controlled by the agent
    
    \( G: \) requirement (software agent) or assumption (environment agent)

[Damas et al, ACM FSE’2006]

Inferring Maintain goals: general idea

- Invariant on all given scenarios:
  
  \(?\text{Inv} = \Box (S0 \lor S1 \lor S2 \lor S3 \lor S4 \lor S5)\)

- Generalization: \([]\text{Inv}\) holds for any system behavior ?
  to be validated by user
  
  \(\Rightarrow\) simplification (CNF) in shorter goals for readability
Inferring Maintain Goals (2)

- For each agent:
  - Generate decorations along agent's timeline (controlled fluents only)
  - Take disjunction of all these state predicates
    \( ? (sp_1 \lor sp_2 \lor sp_3 \lor \ldots) \)
  - Normalize in CNF form
  - Each clause in CNF is a goal
  - Show the user for validation

Generate agent's fluent decorations

- Controlled fluents for train controller
  - Moving = \(<\text{start}, \text{stop, emergency stop}>\) initially false
  - DoorsClosed = \(<\text{close doors}, \text{open doors, emergency open}>\) initially true
Generate agent’s fluent decorations (3)

Moving = \{\text{start}, \text{stop, emergency stop}\} \text{ initially false}

DoorsClosed = \{\text{close doors}, \text{open doors, emergency open}\} \text{ initially true}

Take disjunction of all the state predicates

\[ \square (\neg \text{Moving} \land \text{DoorsClosed}) \lor (\text{Moving} \land \text{DoorsClosed}) \lor (\text{Moving} \land \text{DoorsClosed}) \lor (\neg \text{Moving} \land \text{DoorsClosed}) \lor \ldots ] \]
Normalize in CNF form

\( (\neg \text{Moving} \land \text{DoorsClosed}) \lor (\text{Moving} \land \text{DoorsClosed}) \lor \\
(\text{Moving} \land \text{DoorsClosed}) \lor \ldots \lor (\neg \text{Moving} \land \text{DoorsClosed}) \lor (\ldots) \)

\( (\neg \text{Moving} \lor \text{DoorsClosed}) \)

One goal inferred in this case, shown in alternative ways:

\( (\neg \text{Moving} \lor \text{DoorsClosed}) \)
\( (\text{Moving} \rightarrow \text{DoorsClosed}) \)
\( (\neg \text{DoorsClosed} \rightarrow \neg \text{Moving}) \)
\( \neg (\neg \text{DoorsClosed} \land \text{Moving}) \)

Goal validation by user

- If goal is accepted:
  added to spec and constrains induction

- If goal is rejected:
  user asked to provide scenario counterexample 
  to enrich collection of negative scenarios
Inferring Immediate Response Goals

- **Input**
  - positive scenarios
  - definition of fluents

- **Output**
  - for each agent, a list of goals \( G_i = \Box (A \land o B) \)
    - \( A \): conjunction of monitored/controlled fluents
    - \( B \): disjunction of controlled fluents
    (requirements for software agents or assumptions for environment agents)

- **Problem**
  - not closed under stuttering
  - but gives useful information about single agent stimuli/responses

#### Similar to Maintain goals

The formula to put in CNF is:

\[
\Box ((sp_1 \land o cp_2) \lor (sp_2 \land o cp_3) \lor \ldots)
\]

- \( sp \): state predicate
- \( cp \): controlled predicate

**Example**

\[
\Box (!\, Moving \land \neg DoorsClosed \land \neg Alarmed \land o \, Moving \\
    \land \neg o \, DoorkSeclosed \\
    \lor !\, Moving \land !\, DoorkSeclosed \land \neg Alarmed \land o \, Moving \\
    \land o \, DoorkSeclosed) \lor \ldots
\]
Immediate Response Goals for train controller

- **CNF form obtained**: 
  \[ (\neg \text{alarmed} \lor \neg \text{moving}) \land (\neg \text{alarmed} \lor \neg \text{moving} \lor \neg \text{doorsOpen}) \]
  \[ \land (\neg \text{doorsOpen} \lor \neg \text{moving}) \land (\neg \text{moving} \lor \neg \text{doorsOpen}) \]
  \[ \land (\neg \text{doorsOpen} \lor \neg \text{doorsOpen}) \]

- **Five goals proposed to user**: 
  \[ (\text{alarmed} \rightarrow \neg \text{moving}) \]
  \[ (\text{alarmed} \lor \neg \text{moving} \rightarrow \neg \text{doorsOpen}) \]
  \[ (\text{doorsOpen} \rightarrow \neg \text{moving}) \]
  \[ (\text{moving} \rightarrow \neg \text{doorsOpen}) \]
  \[ (\text{doorsOpen} \rightarrow \neg \text{doorsOpen}) \]

Back to the WinWin partnership ...

- **Goal inference yields drastic improvement of LTS synthesizer**
  on bigger example ... (5 fluents)
  - without goals: 9 questions
    (4 positive, 5 negative)
  - with goals inferred by tool: 4 questions
    (4 positive, 0 negative)

- The goal - scenario - stateMachine triangle is a rich, not yet fully exploited, source of synergy and mutual reinforcement for RE techniques
Further reading on goal-based reasoning


Further reading on goal-based reasoning (2)


Conclusion

- It is important to verify that your software implements its specs correctly... but ...
- ... are those specs meeting the software requirements (including non-functional ones)?
- ... are those requirements meeting the system’s goals? ... under realistic assumptions?
- ... are such goals, requirements & assumptions complete, adequate, consistent?

this is a critical though still largely unexplored area with many challenging issues for formal methods

Conclusion (2)

- Rich models are essential to support the RE process
  - multiple dimensions: intentional, structural, responsibility, operational, behavioral
  - software + environment (e.g., humans, devices, other software, mother Nature, attacker, attackee)
    - start thinking about high assurance at RE time
  - alternative refinements, assignments, resolutions
  - seamless transition from high-level concerns to operational requirements
What models?

Goals

why? how?

Agents, responsibilities

who?

Objects

on what?

Operations

what?

What models?

Hazards

Interaction scenarios

Threats

Behaviors
Conclusion (3)

- The building of such models is hard & critical; should therefore be guided by methods...
  - systematic
  - top-down + bottom-up
  - incremental
  - supporting the analysis of partial models

Conclusion (4)

- Goal-based reasoning is central to RE for...
  - model building & elaboration of requirements
  - exploration & evaluation of alternatives
  - conflict management
  - anticipation of bad behaviors
    (requirements-level exception handling)
  - optimization of model synthesis
Conclusion (5)

♦ Goal completeness can be achieved through multiple means ...
   - refinement checking => missing subgoals
   - obstacle/threat analysis => countermeasure goals
   - animation

Conclusion (6)

♦ Importance of declarative specs: for ...
   communicating with decision makers
   reasoning about models
   optimizing model synthesis

♦ Uniform framework integrating ...
   - current system, system-to-be, future evolutions
     = alternative subtrees in goal AND/OR graph
   - different sub-models for different views
Conclusion (7)

- Be pessimistic from beginning about software and environment hazards, threats, conflicts

- Benefits of multi-button framework
  - semi-formal ...
    for modeling, navigation, traceability
  - formal, when and where needed ...
    for precise, incremental reasoning on model pieces

Goal-oriented models offer lots of opportunities for formal methods

Thanks ...

- to the KAOS crew at UCL, CEDITI and CETIC as researchers, consultants, or tool developers
  C. Damas, A. Dardenne, R. Darimont,
  R. De Landtsheer, E. Delor, D. Janssens,
  B. Lambeau, E. Letier,
  P. Massonet, C. Ponsard, A. Rifaut, H. Tran Van

- to Steve Fickas and his group at Univ. Oregon

- to the EU & Region of Wallonia for significant funding of those efforts
The Objectiver tool

KAOS model browser

KAOS model editor + model query system

Requirements documents generation

More information ...

◆ ... on the method & associated techniques


www.info.ucl.ac.be/~avl

◆ ... on the tools:

http://objectiver.com
http://faust.etic.be