Goal-oriented requirements engineering: the KAOS approach

Ilias Gerostathopoulos

iliasg@d3s.mff.cuni.cz
**Goal**: To model, analyze & generate skeletons of application in DEECo
Reminder: Predicate model

Captures both:
1. requirements
2. architecture

Leaf predicates mapped to component
- communication
- computational semantics
What is Requirements Engineering?

_Broadly_, the process of discovering the purpose a (software) system is intended for.

**Why lies within the scope of RE?**
To identify, analyze and agree on
a) _what_ problem should be solved
b) _why_ it needs to be solved
c) _who_ should be involved in the responsibility of solving the problem

**Why is RE important?**
**Measure of success** of a software system: the degree to which the system satisfies its purpose

**What is the outcome of requirements analysis?**
Typically, a requirements document for the system-to-be (SRS)
What makes RE difficult?

Multiple Transitions to handle

- informal problem world → formal machine world
- high-level, strategic → low-level, technical
- imprecise, unstructured → precise, structured
- implicit, hidden → explicit, adequate
- conflicting → consistent
- partial → sufficiently complete
- intended, ideal → unexpected, realistic
Goal-oriented model-building at RE time

Goal-orientation enables:

• early, incremental analysis
• completeness and pertinence of the model
• reasoning about alternative options
• validation by stakeholders
• backward traceability

Thinking about goals in the early phases of software development is a natural thing; in GORE it is just made explicit
Approaches in GORE

• KAOS
  “a GORE approach with a rich set of formal analysis techniques”
  → Lamsveerde et al.

• i*
  “an agent-oriented modeling framework that can be used for requirements engineering”
  → Eric Yu

• TROPOS
  “an agent-oriented software engineering methodology”
  → John Mylopoulos et al.
KAOS multi-view modeling

Goal model

- SafeTransportation
  - SpeedLimited
  - NoCollision
  - DoorsClosedWhileMoving

Risk model

- NoStopAtSignal

Concern:

- Train
  - On: 0..1
  - Block
- Platform

Responsibility:

- TrackingSystem: Speed, Position
- TrainController

Operationalization:

- Compute Acceleration
- Send Acceleration

Operation model:

- TrainController

Coverage:

- Controller
  - Stop
  - Opening
  - Closing
  - Start
- Passenger

Behavior model:

- Doors
  - opening [AtPlatform]: Closed
  - [TimeOut]: Open
- Movement
  - start [doorsClosed]: Stopped
  - Moving
Goal model - I

**behavioral (hard) goals:**
- intended behaviors
- clear-cut criterion of satisfaction
→ Operational models

**functional goals:**
Underlying operation, feature, service, task

**non-functional goals:**
quality goals e.g. security, accuracy, architectural, development goals, etc

**soft goals:**
- preferred behaviors
- no clear-cut criterion of satisfaction
→ Alternative options

Goal-oriented requirements engineering: the KAOS approach

Ilias Gerostathopoulos, D3S Seminar, 12.12.2012
Goal model - II

alternative refinement (OR-decomposition)

alternative refinement (OR-decomposition)

requirement

domain property

assumption

agents

Vehicle

Vehicle

Station Operator

Achieve[Lot in station booked if available]

Achieve[Batteries charged in stations]

Maintain[Vehicle batteries charged]

Avoid[Vehicle out of battery]

Achieve[Alarm issued when battery low]

Achieve[Alarm handled when issued]

Charging stations available

Charging stations operational

Vehicle

Goal-oriented requirements engineering: the KAOS approach
Name Lot in Station booked if available
Def If a place is available, then it must be booked by the vehicle in order to recharge
Type Achieve
Category Satisfaction
Source interview with VW
Priority Medium

**FormalSpec**
\[
\forall v: \text{Vehicle}, cl: \text{ChargingLot}:
\text{LowBattery}(v) \land \text{Available}(cl) \Rightarrow \\
\Diamond_{\leq T} \text{Booked}(v, cl)
\]

**Real-time linear temporal logic:** $\circ P, \Diamond P, P \cup N, P \land N, \text{and operators on past}$
Object model

Objects: Entities, Associations, Events

Structure/Object model: UML class diagram notation

*Only objects concerned in/referenced by goals are described*
Operations model

**Operation** BookChargingLot

**Def** If a place is available, then it must be booked by the vehicle in order to recharge

**Input** cl: ChargingLot, v: Vehicle

**Output** cl

**DomPre** cl.available = true

**DomPost** cl.available = false

**ReqPre ...**

**ReqTrig for** LotInStationBookedIfAvailable:

LowBattery(v) \(\land\) Close(v, cl)

**ReqPost ...**

DomPre, DomPost: what the operation means in the domain

ReqPre, ReqTrig, ReqPost: additional strengthening to ensure the associated goal
**What Goals provide in KAOS**

*sufficient completeness criterion:*

A specification is complete with respect to a set of goals if all the goals can be proven to be achieved from the specification and the properties known about the domain.

*pertinence criterion:*

A requirement is pertinent with respect to a set of goals if its specification is used in the proof of at least one goal.
Goals refinement checking

A refinement of goal G into subgoals $SG_1, ..., SG_n$ is correct, when it is

- complete: $\{SG_1, ..., SG_n, DOM\} \models G$
- consistent: $\{SG_1, ..., SG_n, DOM\} \not\models false$
- minimal: $\{SG_1, ..., SG_{j-1}, SG_{j+1}, ..., SG_n, DOM\} \not\models G$

How to check goal refinements?

1. Use LTL theorem prover
   - heavyweight, nonconstructive
2. Use bounded SAT solver
   - input: $SG_1 \land ... \land SG_n \land Dom \land \neg G$
   - incremental check/debug
3. Reuse refinement patterns
Refinement patterns - I

- Catalogue of patterns encoding *refinement tactics*
- Generic refinements proved formally, once for all
- Reuse through instantiation, in matching situation

Examples:

```
C \implies \Box T

C \land \text{Case1} \implies \Box T1
C \land \text{Case2} \implies \Box T2
\neg (\text{Case1} \land \text{Case2})
T1 \lor T2 \implies T
```
Refinement patterns - II

Guard introduction refinement

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))

Maintain [TrainWaiting]
On (tr, b) ⇒ On (tr, b) W On (tr, next(b))

Ilias Gerostathopoulos, D3S Seminar, 12.12.2012
Apart from goal refinement, patterns can be applied:

- **Goal operationalization**
- **Obstacle analysis**
Advantages of KAOS:
1. Formal methods *when* and *where* needed (2-button approach)
2. Decomposition patterns
3. Wide system perspective
4. Rich meta-model

Main **difference** from our approach:
1. Different outcome: KAOS framework not intended for code generation