KAOS Construct Analysis using the UEML Approach Template
Matulevicius, Raimundas; Heymans, Patrick

Publication date: 2005

Citation for published version (HARVARD):
Matulevicius, R & Heymans, P 2005, KAOS Construct Analysis using the UEML Approach Template.
KAOS Construct Analysis using the UEML Approach Template

(Technical report)

Raimundas Matulevičius
and Patrick Heymans

InterOP

Namur, Belgium
2005
KAOS Construct Analysis using the UEML Approach Template
Outline

KAOS : Achieve goal .......................................................................................................................... 5
KAOS : Agent ...................................................................................................................................... 9
KAOS : Assignment .......................................................................................................................... 13
KAOS : Avoid goal ............................................................................................................................ 17
KAOS : Boundary condition .............................................................................................................. 21
KAOS : Cease goal ............................................................................................................................ 25
KAOS : Conflict ................................................................................................................................ 29
KAOS : Control .................................................................................................................................. 33
KAOS : Domain property .................................................................................................................. 35
KAOS : Environment agent ................................................................................................................ 39
KAOS : Event ...................................................................................................................................... 41
KAOS : Expectation .......................................................................................................................... 43
KAOS : Goal ....................................................................................................................................... 47
KAOS : Goal refinement .................................................................................................................... 51
KAOS : Input ...................................................................................................................................... 55
KAOS : Maintain goal ........................................................................................................................ 59
KAOS : Monitor .................................................................................................................................. 63
KAOS : Object .................................................................................................................................... 65
KAOS : Operation ............................................................................................................................... 69
KAOS : Operationalisation ................................................................................................................ 73
KAOS : Output .................................................................................................................................... 79
KAOS : Performance .......................................................................................................................... 83
KAOS : Requirement .......................................................................................................................... 87
KAOS : Softgoal ................................................................................................................................. 91
KAOS : Software agent ...................................................................................................................... 93
KAOS Construct Analysis using the UEML Approach Template
KAOS Construct Analysis using
the UEML Approach Template

KAOS : Achieve goal

<table>
<thead>
<tr>
<th>Document type:</th>
<th>Working document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain/Task/Topic:</td>
<td>DEM / UEML / Approaches</td>
</tr>
<tr>
<td>Version:</td>
<td></td>
</tr>
<tr>
<td>Date:</td>
<td>2005.11.30</td>
</tr>
<tr>
<td>Status:</td>
<td>2nd iteration</td>
</tr>
<tr>
<td>Authors:</td>
<td>Raimundas Matulevičius</td>
</tr>
<tr>
<td>Distribution list:</td>
<td>DEM</td>
</tr>
<tr>
<td>Document history:</td>
<td></td>
</tr>
</tbody>
</table>

Achieve goals are goals requiring that some property eventually holds.
Achieve goals state that some target condition should hold in some (bounded) future state.

1. Preamble

Builds on
Goal

Built on by

Construct name
Achieve goal

Alternative construct names
goal, requiring that a property eventually holds

Related, but distinct construct names
goal with a pattern achieve

Related terms
Goal: a prescriptive assertion capturing some objective to be met by cooperation of agents from the agent model.
Maintain goal: a goal requiring that some property always holds.
Avoid goal: a goal requiring that some property never holds.
Cease goal: a goal requiring that some property eventually stops to hold.

Softgoal: goal that do not have a clear-cut criterion for their satisfaction.
Terminal goals: a goal which has no G-refinement.
Requirement: a goal assigned to an agent in the software to be.
Expectation (assumption): a goal assigned to an agent in the environment.
Comment: Can a goal with a pattern be softgoal, terminal goal (requirement and expectation)?

Language

Diagram type
Goal model
2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes

Relations to other constructs

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Both type and instance level

Classes of things

1:1, “goalOwner” played by StakeholderThing.
    Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
    StakeholderThing is subclass of the BWW-HumanThing.

1:1, “concernedObject” played by AllThing.
    Describing object concerned by a goal.

Comment: Achieve goal has the same classes as a goal. It belongs to a goal owner and has the sub-properties. These classes are described here as they are used in property definition.

Properties (and relationships)

1:1, “theAchieveGoal” played by ComplexTransformationLaw.
    Belongs to: 0:1 [1:1], goalOwner.
    Transformation law: a change is required between a state where the concerned object properties are false and one where they are true.
    Representing the achieve goal which is held by a goal owner and requires that some concerned object properties eventually hold.

Comment: For more achieve goal properties (such as concExplicitObjAttribute, concImplicitObjAttribute, attributeName, attributeDef, attributeFormalSpec, attributePriority, and attributeCategory), see goal.
Behaviour
   Existence

Modality (permission, recommendation etc)
   Intention of a goal owner;

4. Open Issues

Change List
KAOS Construct Analysis using the UEML Approach Template
An agent is an active object (or “processor”) which plays a specific role towards goal achievement by controlling specific object behaviours. The focus is thus on a specific role rather than a specific individual.

Agents are active objects, that is, they are capable of performing operations.

1. Preamble

Builds on

Object

Built on by

Environment agent
Software agent

Construct name

Agent

Alternative construct names

An active object
A processor

Related, but distinct construct names

- Environment agent: e.g., pre-existing software component, sensor, actuator, human, organizational unit, etc.
- Software agent: an agent in the system-to-be.

Related terms

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Agent model

2. Presentation

Builds on
Built on by
- Assignment,
- Performance,
- Controls,
- Monitors

Icon, line style, text

User-definable attributes

1:1 \textit{Name}: String = "". A string allowing for unambiguous reference to corresponding instances at the application level.

1:1 \textit{Def}: FreeText = "". Free text used for precise, unambiguous definition of the corresponding instances at the application level.

Relations to other constructs
- Belongs 1:1 to agent model.
- 1:n [1:1] \textit{responsibleAgent} : assignment. Agent has assignment to satisfy the goal.

\textbf{Comment}: Agents are objects. This means that agent could also not be a responsible agent as they could be defined in the object hierarchy.

- 0:n [1:n] \textit{performs} : operation. Agent performs operation in order to satisfy operationalised by this operation goal, which is assigned to this agent.
- 0:n [0:n] \textit{monitors} : object. Agent monitors ("reads") the attribute of the object.
- 0:n [0:n] \textit{controls} : object. Agent controls ("writes") the attribute of the object.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level
- Both type and instance level

Classes of things

1:1 "theAgent" played by \textit{ActiveComponentThing}. Represents the agent.

1:1 "monitoredControlledObject" played by \textit{ComponentThing}. Represents object, controlled or monitored by an agent.

Properties (and relationships)

1:1 [1:1], "attributeName" played by \textit{AnyRegularProperty}.
- \textbf{Belongs to}: theAgent.
- Represents agent attribute name.
1:1 \([1:1]\), \("attributeDef\) played by \(AnyRegularProperty\).

- **Belongs to:** \(theAgent\).
- Represents agent attribute def.

1:n \([1:n]\), \("monitoredImplicitObjAttribute\) played by \(EmergentBindingMutualProperty\)

- **Belongs to:** \(0:n \,[0:n]\), \(monitoredControlledObject\)
- **Belongs to:** \(0:n \,[0:n]\), \(theAgent\)

An agent monitors an object, without defining the concrete attribute of the control.

Also represented: Monitors.

1:n \([1:n]\), \("monitoredExplicitObjAttribute\) played by \(EmergentBindingMutualProperty\)

- **Belongs to:** \(0:n \,[0:n]\), \(monitoredControlledObject\)
- **Belongs to:** \(0:n \,[0:n]\), \(theAgent\)

An agent monitors an object attribute.

Also represented: Monitors.

1:n \([1:n]\), \("controlledImplicitObjAttribute\) played by \(EmergentBindingMutualProperty\)

- **Belongs to:** \(0:n \,[0:n]\), \(monitoredControlledObject\)
- **Belongs to:** \(0:1 \,[0:n]\), \(theAgent\)

An agent controls an object, without defining the concrete attribute of the control.

Also represented: Controls.

1:n \([1:n]\), \("controlledExplicitObjAttribute\) played by \(EmergentBindingMutualProperty\)

- **Belongs to:** \(0:n \,[0:n]\), \(monitoredControlledObject\)
- **Belongs to:** \(0:1 \,[0:n]\), \(theAgent\)

An agent controls an object attribute.

Also represented: Controls.

**Behaviour**

Existence

**Modality (permission, recommendation etc)**

Regular assertion

4. Open Issues

TBF – Dependency constraint between agents as through goal or through operation.

TBF – A goal defines a set of admissible histories in the composed system. Intuitively, a history is a temporal sequence of states of the system. Specify Scenario, Snapshot, Interaction, Source, Target, and State transition constraints. This is related to Agent, Event, and Operation constraints.
KAOS Construct Analysis using the UEML Approach Template
The Assignment is introduced as target of an OR-Assignment meta-relationship from Goal to capture alternative assignments of the same terminal goal to different agents; alternative assignments result in different system proposals in which more or less is automated.

1. Preamble

Builds on

Built on by

Construct name

Assignment

Alternative construct names

Responsibility assignment

Related, but distinct construct names

Related terms

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Agent model

2. Presentation

Builds on

Built on by
**Icon, line style, text**

![Diagram of KAOS construct analysis using the UEML approach template]

**User-definable attributes**

\[
[0:1] \text{AltName : String = "". Name of alternative OR-assignments.}
\]

**Relations to other constructs**

- Belongs to 1..1 agent model.
- 1:1 [0:n], \text{assignedGoal}: goal. A goal, if it is a terminal goal, could be assigned.
- 1:1 [1:n], \text{responsibleAgent}: agent. Responsible agent (software agent or environment agent) is responsible for goal satisfaction.

**Diagram layout conventions**

**Other usage conventions**

**3. Representation**

**Builds on**

- Goal
- Agent

**Built on by**

**Instantiation level**

- Instance and type level

**Classes of things**

1:1 \text{"responsibleAgent" played by ActiveComponentThing.}

- Represents the responsible agent.

1:1, \text{"goalOwner" played by StakeholderThing.}

- Describing \text{goal owner} which holds the goal. Class \text{StakeholderThing} has a characteristic \text{holds goal.}

\text{StakeholderThing} is subclass of the BWW-HumanThing.
Properties (and relationships)

1:1, "theGoal" played by ComplexLawProperty.
   **Belongs to:** 0:1 [1:1], goalOwner.
   **Law:** restricts the possible values of the object attributes.
   Representing the goal which is held by a goal owner.

1:1, "terminalGoal" played by StateLaw.
   **Belongs to:** 0:1 [1:1], goalOwner.
   **Sub-property:** 1:1 [1:1], theGoal.
   **State law:** ∀g ∈ Goal, ∀a ∈ Assignment, a.assignedGoal = g
   \(⇒ \neg ∃ gr ∈ G-refinement: gr.superGoal = g\)
   Only terminal goals can be assigned.

1:1, "theAssignment" played by ComplexBindingMutualProperty.
   **Type:** OR relationship.
   **Belongs to:** 1:1 [1:n] responsibleAgent.
   **Sub-property:** 1:1 [0:n] terminalGoal.
   Describing the assignment.

0:1, "attributeAltName" played by AnyRegularProperty.
   **Sub-property:** theAssignment.
   Represents assignment attribute altName.

Behaviour

Existence

Modality (permission, recommendation etc)

Regular assertion

4. Open Issues
Avoid goals are goals requiring that some property never holds.
Avoid goals state that some target condition on system states should never hold under some current condition.

1. Preamble

Builds on
Goal

Built on by

Construct name
Avoid goal

Alternative construct names
goal, requiring that some property never holds

Related, but distinct construct names
goal with a pattern avoid

Related terms

Goal: a prescriptive assertion capturing some objective to be met by cooperation of agents from the agent model.
Maintain goal: a goal requiring that some property always holds.
Cease goal: a goal requiring that some property eventually stops to hold.
Achieve goal: a goal requiring that some property eventually hold.

Softgoal: goal that do not have a clear-cut criterion for their satisfaction.
Terminal goals: a goal which has no G-refinement.
Requirement: a goal assigned to an agent in the software to be.
Expectation (assumption): a goal assigned to an agent in the environment.

Comment: Can a goal with a pattern be softgoal, terminal goal (requirement and expectation)?

Language

Diagram type
Goal model
2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes

Relations to other constructs

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Both type and instance level

Classes of things

1:1, “goalOwner” played by StakeholderThing.
   Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
   StakeholderThing is subclass of the BWW-HumanThing.

1:1, “concernedObject” played by AllThing.
   Describing object concerned by a goal.

Comment: Avoid goal has the same classes as a goal. It belongs to a goal owner and has the sub-properties. These classes are described here as they are used in property definition.

Properties (and relationships)

1:1, “theAvoidGoal” played by ComplexStateLaw.
   Belongs to: 0:1 [1:1], goalOwner.
   State law: indicates states that cannot be in concerned object.
   Representing the avoid goal which is held by a goal owner and requires some properties of the concerned object never holds.

Comment: For more avoid goal properties (such as concExplicitObjAttribute, concImplicitObjAttribute, attributeName, attributeDef, attributeFormalSpec, attributePriority, and attributeCategory), see goal.
Behaviour
  Existence

Modality (permission, recommendation etc)
  Intention of a goal owner;

4. Open Issues

Change List
KAOS Construct Analysis using
the UEML Approach Template
KAOS : Boundary condition

Boundary condition describes inconsistencies in the considered domain – this means that two or more different goals could not be achieved together.

1. Preamble

Builds on

Built on by

Construct name

Boundary condition

Alternative construct names

Inconsistencies in the considered domain

Related, but distinct construct names

Related terms

Conflict

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Goal model

2. Presentation

Builds on

Built on by

Icon, line style, text

Boundary condition
User-definable attributes
[1:1] **Name**: String = "". A string allowing for unambiguous reference to corresponding instances at the application level.

[1:1] **Def**: FreeText = "". Free text used for precise, unambiguous definition of the corresponding instances at the application level.

[0:1] **FormalSpec**: KAOS real time temporal logic expression. Its values at the application level specify the corresponding **Def** attribute in the KAOS real-time temporal logic.

[0:1] **Likelihood**: propability ∈ [0..1]. Its values at the application level specify how likely the boundary condition is.

[0:1] **Criticality**: set_of{critical, ..., not critical}. Its values at the application level specify how severe the consequences of the resulting conflict are.

Relations to other constructs
Belongs 1:1 to goal model.

1:1 [1:1], **existUnder**: conflict. Conflict exist only under some boundary condition.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on
Conflict

Built on by

Instantiation level
Instance level
Comment: can we define classes of boundary conditions?

Classes of things

Properties (and relationships)
1:1, “theBoundaryCondition” played by StateLaw.
   Type: Boolean, default value: true.
   Sub-property: 1:1 [1:1] **theConflict**.
   State law: two (or more) goals in the same G-refinement cannot be satisfied together.
   Describing boundary condition.

1:1 [1:1], “attributeName” played by AnyRegularProperty.
   Sub-property: **theBoundaryCondition**.
   Represents boundary condition attribute name.

1:1 [1:1], “attributeDef” played by AnyRegularProperty.
   Sub-property: **theBoundaryCondition**.
   Represents boundary condition attribute def.

1:1 [1:1], “attributeFormalSpec” played by AnyRegularProperty.
   Sub-property: **theBoundaryCondition**.
   Represents boundary condition attribute formalSpec.

0:1 [1:1], “attributeLikelihood” played by AnyRegularProperty.
   Sub-property: **theBoundaryCondition**.
   Represents boundary condition attribute likelihood.
0:1 [1:1], “attributeCritically” played by AnyRegularProperty.
Sub-property: theBoundaryCondition.
Represents boundary condition attribute critically.

Behaviour
State

“logicalInconsistency” played by unstableState.
Defining property: theBoundaryCondition,
State constraint: Two or more different goals could not be achieved together.

Modality (permission, recommendation etc)
Regular assertion

4. Open Issues
TBD – describe state law in a more formal way.
KAOS : Cease goal

Cease goals are goals requiring that some property eventually stops to hold.
Cease goals state that some target condition should not hold in some (bounded) future state.

1. Preamble

Builds on

Goal

Built on by

Construct name

Cease goal

Alternative construct names

goal, requiring that some property eventually stops to hold.

Related, but distinct construct names

goal with a pattern cease

Related terms

Goal: a prescriptive assertion capturing some objective to be met by cooperation of agents from the agent model.
Maintain goal: a goal requiring that some property always holds.
Avoid goal: a goal requiring that some property never holds.
Achieve goal: a goal requiring that some property eventually hold.

Softgoal: goal that do not have a clear-cut criterion for their satisfaction.
Terminal goals: a goal which has no G-refinement.
Requirement: a goal assigned to an agent in the software to be.
Expectation (assumption): a goal assigned to an agent in the environment.
Comment: Can a goal with a pattern be softgoal, terminal goal (requirement and expectation)?

Language


Diagram type

Goal model
2. Presentation

Builds on
Built on by
Icon, line style, text

User-definable attributes

Relations to other constructs
Diagram layout conventions
Other usage conventions

3. Representation

Builds on
Built on by

Instantiation level
Both type and instance level

Classes of things
1:1, “goalOwner” played by StakeholderThing.
Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
StakeholderThing is subclass of the BWW-HumanThing.

1:1, “concernedObject” played by AllThing.
Describing object concerned by a goal.

Comment: Cease goal has the same classes as a goal. It belongs to a goal owner and has the sub-properties. These classes are described here as they are used in property definition.

Properties (and relationships)
1:1, “theCeaseGoal” played by ComplexTransformationLaw.
Belongs to: 0:1 [1:1], goalOwner.
Transformation law: a change is required between a state where the concerned object properties are true and one where they are false.
Representing the cease goal which is held by a goal owner and requires that some concerned object properties eventually stops to hold.

Comment: For more cease goal properties (such as concExplicitObjAttribute, concImplicitObjAttribute, attributeName, attributeDef, attributeFormalSpec, attributePriority, and attributeCategoty), see goal.

Behaviour
Existence
**Modality (permission, recommendation etc)**

*Intention of a goal owner;*

4. Open Issues

Change List
Two or more goals are considered to be conflicting when under some boundary condition the goals become logically inconsistent in the considered domain – these goals could not be achieved together. Goals $G_1, G_2, \ldots, G_n$ are said to be conflicting (or “divergent”) if under some boundary condition the goals become logically inconsistent in the domain considered, that is, they cannot be achieved altogether.

1. Preamble

Builds on

Built on by

Construct name

Conflict

Alternative construct names

Conflicting goals
Divergent goals

Related, but distinct construct names

Related terms

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Goal model

2. Presentation

Builds on

Built on by

Icon, line style, text

Conflict
User-definable attributes

Relations to other constructs

2:n [0:n], conflictBetweenGoals : goals.
0:n [0:n], isInDomain : domain properties.
1:1 [1:1], existUnder : boundary condition.

Diagram layout conventions
Cross between conflicting goals is represented in red.

Other usage conventions

3. Representation

Builds on
Two or more goals

Built on by
Boundary condition
Domain property

Instantiation level
Instance level
Comment: can we define classes of conflicts?

Classes of things
1:1, “goalOwner” played by StakeholderThing.
Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
StakeholderThing is subclass of the BWW-HumanThing (specified in goal template).
If goals are conflicting, this means conflict between the goal owners.

1:1, “concernedObject” played by ComponentThing.
Describing object concerned by a goal. This object is characterised by a domain property.

Properties (and relationships)
1:1, theConflict played by MutualProperty. Describing the conflict.
Sub-property of 2:n [0:n] theGoal.
Sub-property of 0:n [0:n] domainHypothesis.
Sub-property of 0:n [0:n] domainInvariant.
Conflict specifies mutual property between two or more goal owners (we consider what two conflicting goals have different goal owners).

Comment: theGoal property is specified in the template for the goal construct.

1:1 [1:1], boundaryCondition played by StateLaw
Sub-property of 1:1 [1:1] theConflict.
State law: Conflict exist under some boundary condition.
The conflict exists only if some boundary condition, which defines why two or more goals can not be satisfied together exists.
Also represented by: boundary condition.

0:n [0:n], domainHypothesis played by AnyProperty.
Conflicts are described in the domain which is specified by the domain properties.
Also represented by: domain property.
0:n, domainInvariant played by AnyProperty.
Conflicts are described in the domain which is specified by the domain properties.
Also represented by: domain property.

**Behaviour**
Regular assertion

**Modality (permission, recommendation etc)**
Obligation of Boundary Condition.

4. Open Issues
KAOS Construct Analysis using the UEML Approach Template
KAOS : Control

Agent controls ("writes") the value of the object attribute.

1. Preamble

Builds on

Built on by

Construct name

Controls

Alternative construct names

 Writes

Related, but distinct construct names

Related terms

 Monitors : Agent monitors ("reads") the value of the attribute.

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Agent model

2. Presentation

Builds on

Built on by

Icon, line style, text

Agent name

Object name

Controls
User-definable attributes

- **WhichAtt**: String = “”.indicate which attributes of the object are specifically controlled.

Relations to other constructs

- Belongs to 1..1 agent model.
- 0:n [1:1], Object. *Object* is controlled by an *agent*.
- 0:n [1:1], Agent. *Agent* controls an *object*.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

- Agent
- Object

Built on by

Instantiation level

- Instance and type level

Classes of things

1:1 “controlledObject” played by ComponentThing. Represents *object*, controlled by an *agent*.

1:1 “controllingAgent” played by ActiveComponentThing. Represents the agent.

Properties (and relationships)

1:1, “theControls” played by BindingMutualProperty.

- Belongs to: 1:1 [0:n], controllingAgent.
  - Describing controls relationship.

1:n, “explicitObjAttribute” played by AnyProperty.

- Belongs to: controlledObject.
  - Sub-property of: theControls.
  - Defines explicitly which attribute of the object is controlled.

1:n, “implicitObjAttribute” played by AnyProperty.

- Belongs to: controlledObject.
  - Sub-property of: theControls.
  - Does not define explicitly which attribute of the object is controlled.

Behaviour

Existence

Modality (permission, recommendation etc)

- Regular assertion

4. Open Issues
KAOS : Domain property

A domain property is a property that is naturally true about the composite system. A domain property (DomProp) is a descriptive assertion about objects in the environment which holds independently of the software-to-be.

1. Preamble

Builds on

Built on by
Domain invariant. A domain invariant is a property known to hold in every state of some domain object. It is an indicative statement of domain knowledge.
Domain hypothesis. A domain hypothesis is a domain property about some domain object supposed to hold and used when arguing about the sufficient completeness of goal refinements.

Construct name
Domain property

Alternative construct names
A property that is naturally true about the composite system
A descriptive assertion about objects in the environment

Related, but distinct construct names

Related terms
- Domain invariant: a property known to hold in every state of some domain object
- Domain hypothesis: a property about some domain object supposed to hold and used when arguing about the sufficient completeness of goal refinements.

Language
KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type
Goal model
Object model
2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes

[1:1], Name = "". A string allowing for unambiguous reference to corresponding instances at the application level.
[1:1], Def: FreeText = "". Free text used for precise, unambiguous definition of the corresponding instances at the application level.
[0:1], FormalSpec: KAOS real time temporal logic expression. Its values at the application level specify the corresponding Def attribute in the KAOS real-time temporal logic.

Relations to other constructs

0:n [0:n], subProperty: goal. Domain properties refine the goal through the G-refinement relationship.
0:n [0:n], isInDomain: Conflict. Conflicts between goals are defined in a domain by domain properties.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Instance level

Classes of things

1:1 “theDomainObject” played by CompositeThing. Describing object to which domain property belongs.

Properties (and relationships)

1:1, “theDomainProperty” played by Anything. Belongs to: theDomainObject. Representing the domain property. Domain property is a property of an object.

1:1 [1:1], “attributeName” played by AnyRegularProperty. Sub-property: theDomainProperty. Represents domain property attribute name.

0:1 [1:1], “attributeFormalSpec” played by AnyRegularProperty.
  Sub-property: theDomainProperty.
  Represents domain property attribute formalSpec.

Behaviour
  Existence

Modality (permission, recommendation etc)
  Regular assertion

4. Open Issues
KAOS: Environment agent

Environment agent (e.g., pre-existing software component, sensor, actuator, human, organizational unit, etc.)

1. Preamble

Builds on
Agent

Built on by

Construct name
Environment agent

Alternative construct names

Related, but distinct construct names

Related terms
- Agents: active objects capable of performing operations.
- Software agent: an agent in the system-to-be.

Language
KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type
Agent model

2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes
Relations to other constructs

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level
  Type level

Classes of things
  1:1, “isEnvironmentAgent” played by ActiveComponentThing.
  Environment agents are agents and inherits all agent attributes and properties.

Properties (and relationships)

Behaviour
  Existence

Modality (permission, recommendation etc)
  Regular assertion

4. Open Issues
Event is an instantaneous object.

1. Preamble

Builds on
Object

Built on by

Construct name
Event

Alternative construct names
An instantaneous object

Related, but distinct construct names

Related terms
Object

Language
KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type
Object model
Operation model

2. Presentation

Builds on

Built on by

Icon, line style, text

\[ \text{EvName} \]

Event
User-definable attributes
Event inherits all the attributes of the object.

Relations to other constructs
Event inherits all the relationships of the object.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level
Instance level

Classes of things
1:1, “theEvent” played by ChangingThing.
Describing the event.

Properties (and relationships)

Behaviour
Event

REPRESENTED STATE ENTRIES

“initialState” played by StateOfAThing
State constraints: State constraints are defined by object implicit and explicit attributes (inputs to the operation).

“resultState” played by StateOfAThing
State constraints: State constraints are defined by object implicit and explicit attributes (outputs from the operation).

REPRESENTED EVENT ENTRIES

“occur” played by EventInAThing
From state: initialState
To state: resultState
Trigger: reqTrig sub-property of operationalisation.
Condition: reqPre and reqPost in operationalisation and domPre and domPost in operation.
Action: when event occurs the operation is caused.

Modality (permission, recommendation etc)
Regular assertion

4. Open Issues
TBF – A goal defines a set of admissible histories in the composed system. Intuitively, a history is a temporal sequence of states of the system. Specify Scenario, Snapshot, Interaction, Source, Target, and State transition constraints. This is related to Agent, Event, and Operation constraints.
KAOS : Expectation

An expectation is a goal assigned to an environment agent.

1. Preamble

Builds on

Goal

Built on by

Construct name

Expectation

Alternative construct names

Assumption

Related, but distinct construct names

Terminal goal: goal which has no G-requirement.

Related terms

Requirement: a goal assigned to an agent in the software-to-be.

Softgoal: a goal that cannot be said to be satisfied in a clearcut sense.

Comment: can an expectation be a softgoal?

Maintain goal: a goal requiring that some property always holds.

Avoid goal: a goal requiring that some property never holds.

Achieve goal: a goal requiring that some property eventually hold.

Cease goal: a goal requiring that some property eventually stops to hold.

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Goal model
2. Presentation

Builds on

Built on by

Icon, line style, text

Expectation

User-definable attributes

Requirement inherits all the attributes of the goal.

Relations to other constructs

- Belongs 1:1 to goal model.
- 1:n [1:n], responsible: environment agent. Expectation is assigned through responsibility relationship to an environment agent.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Comment: Expectation is a goal and has most of the goal classes and properties. But expectation is also a terminal goal, so it has no G-refinement.

Built on by

Instantiation level

Instance level

Comment: Can we have classes of environment agents and these classes or individual expectations assigned?

Classes of things

1:1, “environmentAgent” played by ActiveComponentThing.

Describing the agent an expectation is assigned.

Properties (and relationships)

1:1, “theExpectation” played by ComplexStateLaw.

Belongs to: 0:1 [0:n] environmentAgent.

Belongs to: 0:1 [1:1] goalOwner.

State law: Is restricted by the assignement relationship. An expectation himself restricts state of the concerned object.

Representing the expectation. Expectation as a goal, has a goal owner.

1:1, “isTerminalGoal” played by StateLaw.

Sub-property: 1:1 [1:1], theExpectation.

State law: ∀g ∈ Goal, ∀a ∈ Assignment, a.assignedGoal = g
\[ \Rightarrow \exists \, gr \in G\text{-refinement} : gr.\text{superGoal} = g \]

*Expectation* is a terminal *goal* which means that an *expectation* can not have G-refinement.

**Behaviour**

Existence

**Modality (permission, recommendation etc)**

- *Intention* of a goal owner;
- *Obligation* of an *environment* agent.

**4. Open Issues**
A goal is a prescriptive assertion capturing some objective to be met by cooperation of agents from the agent model. A goal prescribes a set of desired behaviours. A goal defines an objective the composite system should meet usually through the cooperation of multiple agents.

1. Preamble

Builds on

Built on by
- softgoal
- maintain goal
- achieve goal
- cease goal
- avoid goal
- requirement
- expectation (assumption)

Comment: All the mentioned constructs are goals having additional features to the ones defined in this template.

Construct name

Goal

Alternative construct names
- a prescriptive assertion
- a set of desirable behaviours
- an objective a desirable system should meet
- a sub-goal
- a parent goal
- a super goal

Related, but distinct construct names

Related terms
- Softgoal: goal that do not have a clear-cut criterion for their satisfaction.
- Maintain goal: a goal requiring that some property always holds.
- Avoid goal: a goal requiring that some property never holds.
- Achieve goal: a goal requiring that some property eventually hold.
- Cease goal: a goal requiring that some property eventually stops to hold.
- Terminal goals: a goal which has no G-refinement.
- Requirement: a goal assigned to an agent in the software to be.
- Expectation (assumption): a goal assigned to an agent in the environment.
KAOS Construct Analysis using
the UEML Approach Template

Language

Diagram type
Goal model

2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes

[1:1] Name: String. A string allowing for unambiguous reference to a corresponding goal at the application level.

[1:1] Def: String. Free text used for precise, unambiguous definition of the goal at the application level.

[0:1] FormalSpec: KRTTL. Its values at the application level specify the corresponding Def attribute in the KAOS real-time temporal logic.

[0:1] Priority: PriorityType. Values at the application level specify the extent to which the goal is mandatory or optional.

[0:1] Owner: String. Defines stakeholder which identified and argued for that goal.

[0:1] Category: set_of Strings {satisfaction, safety, security, information, accuracy, and others}. Category provides a classification of goals that can be used to guide the acquisition, definition and refinement.

Any other attributes that the user wished to add.

Relations to other constructs

Belongs 1:1 to Goal model.


0:n [0:n] concerns : object. Goal definition refers to the objects and their attributes.

1:1 [0:n] superGoal : goal. Goal is a super (parent) goal in the G-refinement relationship.

0:n [1:n] subGoal : goal. Goal refines a super (parent) goal through the G-refinement relationship.

0:n [0:n] subProperty : domain properties. Super (parent) goals are refined to a subgoal and domain properties through the G-refinement relationship.

0:n [1:1] op_goal : operationalisation. Operationakalisation defines operations which operationalise this goal through required conditions (reqPre, reqTrig, and reqPost).

0:n [2:n] betweenGoals : conflict. One or several goals could be part of the conflict when boundary condition is determined.
Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

- Assignment – defines how goals can be assigned to agents.
- Operationalisation – defines how goals are operationalised.
- G-refinement – defines how goals are refined.
- Conflict – defines the way conflicts between goals are represented.

Instantiation level

- Both type and instance level

Classes of things

1:1, “concernedObject” played by AllThing.
Describing object concerned by a goal.

1:1, “goalOwner” played by StakeholderThing.
Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
StakeholderThing is subclass of the BWW-HumanThing.

Properties (and relationships)

1:1, “theGoal” played by ComplexLawProperty.
Belongs to: 0:1 [1:1] goalOwner.
Law: goal restricts state of the object by concerning it.
Representing the goal which is held by a goal owner.

1:n [1:n], “concExplicitObjAttribute” played by AnyProperty.
Belongs to: 0:n [0:n], concernedObject.
Sub-property: theGoal.
Sub-property: attributeDef.
Sub-property: attributeFormalSpec.
A goal concerns an object’s attribute.

1:n [1:n], “concImplicitObjAttribute” played by AnyProperty.
Belongs to: 0:n [0:n], concernedObject.
Sub-property: theGoal.
Sub-property: attributeDef.
Sub-property: attributeFormalSpec.
A goal concerns an object, without defining the concrete attribute of the concern.

1:1 [1:1], “attributeName” played by AnyRegularProperty.
Sub-property: theGoal.
Represents goal attribute name.

1:1 [1:1], “attributeDef” played by AnyRegularProperty.
Sub-property: theGoal.
Represents goal attribute def.

0:1 [1:1], “attributeFormalSpec” played by AnyRegularProperty.
Sub-property: theGoal.
Represents goal attribute formalSpec.
KAOS Construct Analysis using the UEML Approach Template

0:1 [1:1], “attributePriority” played by AnyRegularProperty.  
Sub-property: theGoal.  
Represents goal attribute priority.

0:1 [1:1], “attributeCategory” played by AnyRegularProperty.  
Sub-property: theGoal.  
Represents goal attribute category.

Behaviour

Existence

Modality (permission, recommendation etc)

Intention of a goal owner;  
Obligation of an agent.

4. Open Issues

In this template it is not considered:  
Obstacle constructs and relationship with a goal.  
Dependency constructs between agents both through goal and/or through operation.  
History (and its constructs) in a composed system.
Goals refinement is a relationship which is used to refine goals to subgoals and to domain properties. Parent goal could have alternative refinements.

1. Preamble

Builds on

OR-refinement
AND-refinement

Built on by

Construct name

Goal refinement

Alternative construct names

Refines
G-refinement

Related, but distinct construct names

Related terms

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Goal model

2. Presentation

Builds on

Built on by
Icon, line style, text

User-definable attributes

[1:1] \textit{Complete}: Boolean. Indicate whether the refinement is \textit{arguably sufficient} (value "complete") or not arguably sufficient (value "undetermined") to satisfy the parent goal.

[0:1] \textit{Tactics}: set of\{IntroduceMilestone, DecomposeAntecedentByCase, IntroduceAccuracyGoal\} document the tactics used for refining the parent goal. Values include IntroduceMilestone, DecomposeAntecedentByCase, IntroduceAccuracyGoal.

[0:1] \textit{AltName}: String ="". To name the corresponding alternative for further reference. In case a goal is refined into multiple alternative G-refinements this meta-attribute is mandatory.

Relations to other constructs

- Belongs to 1..1 Goal model.
- 1:1 [1:1], Goal. \textit{G-refinement} is used to refine a parent goal.
- 1:n [1:1], Goal. \textit{G-refinement} refines parent goal to several subgoals.
- 1:n [1:1], Domain property. \textit{G-refinement} refines parent goal to subgoals and domain properties.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Goal

Built on by

Domain property

Instantiation level

Instance and type levels

Classes of things

1:1, "goalOwner" played by \textit{StakeholderThing}.

Describing \textit{goal owner} which holds the goal. Class \textit{StakeholderThing} has a characteristic \textit{holds goal}.
StakeholderThing is subclass of the BWW-HumanThing (specified in goal template). G-refinement is a mutual relation between goal owners.

1:1, “concernedObject” played by ComponentThing.
Describing object concerned by a goal. This object is characterised by a domain properties – either domain invariant or domain hypothesis.

Properties (and relationships)
1:1, “theGoal” played by ComplexLawProperty.
Belongs to: 0:1 [1:1], goalOwner.
Law: restricts the the possible values of the object attributes.
Representing the goal which is held by a goal owner.

1:1 [0:n], superGoal played by ComplexLawProperty.
Sub-property: 0:1 [1:1] theGoal.
Supergoal is refined by the subgoals. Like a goal, it is complex (has attributes) and law property (has restrictions over the concerned object –see goal template).

1:n [0:n], subGoal played by ComplexLawProperty.
Subgoals refine the super goal. Like goals they are complex (have attributes) and law properties (have restrictions over the concerned object –see goal template).

0:n [0:n], domainHypothesis played by AnyProperty.
A domain hypothesis is a domain property about some domain object supposed to hold and used when arguing about the sufficient completeness of G-refinement.
Also represented by: domain property.

0:n [0:n], domainInvariant played by AnyProperty.
A domain invariant is a property known to hold in every state of some domain object. It is an indicative statement of domain knowledge.
Also represented by: domain property.

1:1, “theG-Refinement” played by ComplexMutualProperty.
Type: AND/OR relationship.
Sub-property: 0:n [1:1] domainHypothesis.
Sub-property: 0:n [1:1] subGoal
Describing goal refinement.

1:1 [1:1], “attributeComplete” played by AnyRegularProperty.
Sub-property: theG-Refinement.
Represents G-refinement attribute complete.

0:1 [1:1], “attributeAltName” played by AnyRegularProperty.
Sub-property: theG-Refinement.
Represents G-refinement attribute altName.

0:1 [1:1], “attributeTactics” played by AnyRegularProperty.
Sub-property: theG-Refinement.
Represents G-refinement attribute tactics.

Behaviour
Existence

Modality (permission, recommendation etc)
Regular assertion
4. Open Issues
An object is among the inputs of an operation if it is among the sorts making up the domain of the relation defined by the operation.

1. Preamble

Builds on

Built on by

Construct name

Inputs

Alternative construct names

Related, but distinct construct names

Related terms

Outputs: An object is among the outputs of an operation if it is among the sorts making up the co-domain of the relation defined by the operation.

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Operation model

2. Presentation

Builds on

Built on by

Icon, line style, text
**User-definable attributes**

[0:1] *WhichAtt* : String = "". Indicate which attributes of the object are specifically taken as input of the operation.

**Relations to other constructs**

- Belongs to 1..1 operation model.
- 0:n [1:1], Object. *Object* is input to an *operation*.
- 0:n [1:1], Operation. *Operation* has input an *object*.

**Diagram layout conventions**

**Other usage conventions**

**3. Representation**

**Builds on**

**Built on by**

**Instantiation level**

**Type level**

**Classes of things**

1:1 “*inObject*” played by *ComponentThing*.
   Represents *object*, which is input to operation.

0:n, “*causingEvent*” played by *ChangingThing*.

**Properties (and relationships)**

1:1, “*theInput*” played by *BindingMutualProperty*.
   Belongs to: 0:n [1:1], *inObject*.
   Describing *input*. Input is a binding mutual property between the event which causes the operation and the object which is input to this operation.

1:n, “*explicitObjAttribute*” played by *AnyProperty*.
   Belongs to: *inObject*.
   Sub-property of: *theInput*.
   Defines explicitly which attribute of the object is an input.

1:n, “*implicitObjAttribute*” played by *AnyProperty*.
   Belongs to: *inObject*.
   Sub-property of: *theInput*.
   Does not define explicitly which attribute of the object is an input.

0:1, “*attributeWhichAtt*” played by *AnyRegularProperty*.
   **Sub-property**: *theInput*.
   Representing the input attribute *whichAtt*.

0:n [1:1], “*forOperation*” played by *TransformationLaw*.
   Belongs to: *causingEvent*.
   Sub-property: *theInput*.
   **Transformation law**: operation gets input.
   Describing the operation which has input.
   Also represented by *operation*.
Behaviour
   Existence

Modality (permission, recommendation etc)
   Regular assertion

4. Open Issues
KAOS Construct Analysis using the UEML Approach Template
Maintain goals are goals requiring that some property always holds
Maintain goals state that some target condition on system states should always hold under some current condition.

1. Preamble

Builds on
Goal

Built on by

Construct name
Maintain goal

Alternative construct names
goal, requiring that some property always holds

Related, but distinct construct names
goal with a pattern maintain

Related terms
Goal: a prescriptive assertion capturing some objective to be met by cooperation of agents from the agent model.
Cease goal: a goal requiring that some property eventually stops to hold.
Achieve goal: a goal requiring that some property eventually hold.
Avoid goal: a goal requiring that some property never holds.

Softgoal: goal that do not have a clear-cut criterion for their satisfaction.
Terminal goals: a goal which has no G-refinement.
Requirement: a goal assigned to an agent in the software to be.
Expectation (assumption): a goal assigned to an agent in the environment.

Comment: Can a goal with a pattern be softgoal, terminal goal (requirement and expectation)?

Language

Diagram type
Goal model
2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes

Relations to other constructs

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Both type and instance level

Classes of things

1:1, “goalOwner” played by StakeholderThing.
   Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
   StakeholderThing is subclass of the BWW-HumanThing.

1:1, “concernedObject” played by AllThing.
   Describing object concerned by a goal.

Comment: Maintain goal has the same classes as a goal. It belongs to a goal owner and has the sub-properties. These classes are described here as they are used in property definition.

Properties (and relationships)

1:1, “theMaintainGoal” played by ComplexStateLaw.
   Belongs to: 0:1 [1:1], goalOwner.
   State law: indicates states that cannot be in concerned object.
   Representing the maintain goal which is held by a goal owner and requires that some properties of the concerned object always holds.

Comment: For more avoid goal properties (such as concExplicitObjAttribute, concImplicitObjAttribute, attributeName, attributeDef, attributeFormalSpec, attributePriority, and attributeCategory), see goal.
**Behaviour**

Existence

**Modality (permission, recommendation etc)**

*Intention* of a goal owner;

4. Open Issues

Change List
KAOS Construct Analysis using the UEML Approach Template
**KAOS : Monitor**

<table>
<thead>
<tr>
<th>Document type:</th>
<th>Working document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain/Task/Topic:</td>
<td>DEM /UEML / Approaches</td>
</tr>
<tr>
<td>Version:</td>
<td></td>
</tr>
<tr>
<td>Date:</td>
<td>2005.11.30</td>
</tr>
<tr>
<td>Status:</td>
<td>2nd iteration</td>
</tr>
<tr>
<td>Authors:</td>
<td>Raimundas Matulevičius</td>
</tr>
<tr>
<td>Distribution list:</td>
<td>DEM</td>
</tr>
<tr>
<td>Document history:</td>
<td></td>
</tr>
</tbody>
</table>

*Agent* monitors (“reads”) the value of the *attribute*.

1. **Preamble**

Builds on

Built on by

**Construct name**

Monitors

**Alternative construct names**

Reads

**Related, but distinct construct names**

**Related terms**

*Controls*: *agent* controls (“writes”) the value of the *object* attribute.

**Language**

KAOS,

http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

**Diagram type**

Agent model

2. **Presentation**

Builds on

Built on by

**Icon, line style, text**

[Diagram of agent and object with an arrow labeled "Monitors"]

**User-definable attributes**

- *WhichAtt*: String = "".indicate which attributes of the object are specifically monitored.
Relations to other constructs

- 0:n [1:1], Object. Object is monitored by an agent.
- 0:n [1:1], Agent. Agent monitors an object.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Instance level

Classes of things

1:1 “monitoredObject” played by ComponentThing.
  Represents object, monitored by agent.

1:1 “monitoringAgent” played by ActiveComponentThing.
  Represents the agent.

Properties (and relationships)

1:1, “theMonitors” played by BindingMutualProperty.
  Belongs to: 1:1 [0:n], monitoringAgent.
  Describing monitors relationship.

1:n, “explicitObjAttribute” played by AnyProperty.
  Belongs to: monitoredObject.
  Sub-property of: theMonitors.
  Defines explicitly which attribute of the object is monitored.

1:n, “implicitObjAttribute” played by AnyProperty.
  Belongs to: monitoredObject.
  Sub-property of: theMonitors.
  Does not define explicitly which attribute of the object is monitored.

Behaviour

Existence

Modality (permission, recommendation etc)

Regular assertion

4. Open Issues
KAOS : Object

An object is a thing of interest in the system being modeled whose instances can be distinctly identified and may evolve from state to state.
An object instance is a thing that can be distinctly identified. A domain-level object describes a set of such instances that share some common characteristics.

1. Preamble

Builds on

Built on by

Construct name

Object

Alternative construct names

a thing of interest
a thing that can be distinctly identified

Related, but distinct construct names

Related terms

Agent
Entity
Event
Relationship

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Object model

2. Presentation

Builds on
Built on by

Icon, line style, text

User-definable attributes

[1:1] Name : String ="". A string allowing for unambiguous reference to corresponding instances at the application level. The name of the object is used to identify the object.

[1:1] Def : Text = "". Tree text used for precise, unambiguous definition of the corresponding instances at the application level. The definition of an object is a natural language statement that should provide a precise interpretation for the set member(Obj), so that one can tell whether or not a particular object instance is currently an instance of the domain-level object.

[0:1] Alive : Boolean = "True/False". Value in some state at the instance level indicates whether or not the corresponding object instance exists in that state, that is, has appeared in the system without disappearing yet.

Relations to other constructs

- Belongs 1:1 to object model.
- 0:n [0:n], concerns : goal. Goals concern objects - this means that their formulation in Def refers to these objects and their attributes.
- 1:n [0:n], input : operation. Operations are related to objects through input links.
- 1:n [0:n], output : operation. Operations are related to objects through output links.
- 0:n [0:n], monitors : agent. An agent monitors an object if the states of the object are directly observable by it.
- 0:n [0:n], controls : agent. An agent controls an object if the states of the object are directly controllable by it.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Both type and instance level

Classes of things

1:1, “theObject” played by ComponentThing.

Properties (and relationships)

1:1 [1:1], “attributeName” played by AnyRegularProperty.
- Sub-property: theGoal.
  Represents goal attribute name.

1:1 [1:1], “attributeDef” played by AnyRegularProperty.
- Sub-property: theGoal.
  Represents goal attribute def.

Behaviour

Existence

Modality (permission, recommendation etc)

Regular assertion
4. Open Issues


**KAOS : Operation**

<table>
<thead>
<tr>
<th>Document type:</th>
<th>Working document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain/Task/Topic:</td>
<td>DEM / UEML / Approaches</td>
</tr>
<tr>
<td>Version:</td>
<td></td>
</tr>
<tr>
<td>Date:</td>
<td>2005.11.30</td>
</tr>
<tr>
<td>Status:</td>
<td>2nd iteration</td>
</tr>
<tr>
<td>Authors:</td>
<td>Raimundas Matulevičius</td>
</tr>
<tr>
<td>Distribution list:</td>
<td>DEM</td>
</tr>
<tr>
<td>Document history:</td>
<td></td>
</tr>
</tbody>
</table>

An operation is an input-output relation over objects; operation applications define state transitions. Operations are characterized by pre-, post-, and trigger conditions. A distinction is made between domain pre-/post- conditions, which capture the elementary state transitions defined by operation applications in the domain, and required pre/trigger/postconditions, which capture additional strengthening to ensure that the goals are met.

1. **Preamble**

- **Builds on**
- **Built on by**

**Construct name**

- Operation

**Alternative construct names**

- An input-output relation

**Related, but distinct construct names**

**Related terms**

**Language**

- KAOS,

**Diagram type**

- Operation model

2. **Presentation**

- **Builds on**
- **Built on by**

**Icon, line style, text**
User-definable attributes

[1:1] **Name**: String = “”. A string allowing for unambiguous reference to corresponding instances at the application level.

[1:1] **Def**: FreeText = “”. Free text used for precise, unambiguous definition of the corresponding instances at the application level.

[0:1] **modifier**: Boolean. Indicate whether the operation is a object **Modifier** or **Observer**.

Relations to other constructs

Belongs 1:1 to goal model.

1:1 [0:n] **operationalise** : goal. **Goals** assigned to agents are **operationalised** through **operations**. The operationalisation of a goal through some operation entails permissions and obligations on the operation’s applications; the latter are captured by the **ReqPre**, **ReqPost** and **ReqTrig** metaattributes of the **Operationalisation** relationship that strengthen the operation’s domain pre/postconditions.

0:n [1:n] **input** : object. **Operations** are related to **objects** through input links.

0:n [1:n] **output** : object. **Operations** are related to **objects** through output links.

1:n [1:1] **occurs** : event. The applications of an **operation** may be caused by **event(s)**. This means that the **operation**’s **ReqTrig** includes a predicate **occurs** on instances of that **event**.

1:n [1:n] **performs** : agent. A meta-model constraint requires any agent **Responsible** for some goal to **Perform** all the **operations** that **Operationalize** that goal in accordance with the permissions and obligations specified in the operation’s **ReqPre**, **ReqTrig** and **ReqPost** conditions. The **Performance** meta-relationship is thus a derived one.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level

Type level

Classes of things

1:n, “**causingEvent**” **played by** **ChangingThing**.
Operation is caused by event(s).

1:1, “**op_object**” **played by** **ComponentThing**.
Operation gets input from and makes output to the object.

Properties (and relationships)

0:n [0:n] “**inputForOperation**” **played by** **BindingMutualProperty**.
**Belongs to**: **op_object**.
Describing input for operation.
**Also represented by**: **Inputs**.

0:n [0:n] “**outputForOperation**” **played by** **BindingMutualProperty**.
**Belongs to**: **op_object**.
Describing output for operation.
**Also represented by**: **Outputs**.

1:1, “**theOperation**” **played by** **TransformationLaw**.
**Belongs to**: **causingEvent**.
**Sub-property of**: **inputForOperation**.
Sub-property of: outputFromOperation.
Transformation law: operation is performed by an agent responsible for the goal fulfilment. Whenever the required conditions hold, performing the operations satisfies the goal.
Representing the operation.

1:1, “attributeName” played by AnyRegularProperty.
Sub-property of: theOperation.
Represents operation attribute name.

1:1, “attributeDef” played by AnyRegularProperty.
Sub-property of: theOperation.
Represents operation attribute altName.

0:1, “modifier” played by AnyRegularProperty.
Sub-property of: theOperation.
Represents operation attribute altName.

Characterising the states before any application of the operation;

Defining a relation between states before and after applications of the operation;

Behaviour
Process

REPRESENTED STATE ENTRIES

“initState” played by StateOfAThing,
Defining property: inputForOperation.
State constraint: implicit and explicit attribute of an object.

“resultState” played by StateOfAThing,
Defining property: outputForOperation
state constraint: implicit and explicit attributes of an object.

REPRESENTED EVENT ENTRIES

“eventOccurs” played by ExternalEvent,
From state: initState
To state: initState
Trigger: reqTrig sub-property of operationalisation.
Condition: reqPre (sub-property of operationalisation) and domPre (sub-property of operation) holds.
Action: operation is initiated,
effected by Event.

“getInput” played by InternalEvent,
From state: initState
To state: initState
Trigger: reqTrig sub-property of operationalisation.
Condition: reqPre (sub-property of operationalisation) and domPre (sub-property of operation) holds.
reqPost (sub-property of operationalisation) and domPost (sub-property of operation) introduced.
Action: object implicit and explicit attributes are taken as the input for the operation.

“setOutput” played by InternalEvent,
From state: initState
To state: resultState
Trigger: reqTrig sub-property of operationalisation.
KLAS Construct Analysis using
the UEML Approach Template

**Condition**: $\text{reqPost}$ (sub-property of $\text{operationalisation}$) and $\text{domPost}$ (sub-property of $\text{operation}$) holds.

**Action**: object explicit and implicit attributes are taken as the input for the operation.

**Modality (permission, recommendation etc)**

Regular assertion

4. Open Issues

TBF – **Dependency** constraint between agents as through goal or through operation.
TBF – A **goal** defines a set of admissible histories in the composed system. Intuitively, a history is a temporal sequence of states of the system. Specify **Scenario**, **Snapshot**, **Interaction**, **Source**, **Target**, and **State transition** constraints. This is related to **Agent**, **Event**, and **Operation** constraints.
The Operationalisation meta-relationship is an AND/OR relationship between goals and required pre, trigger, and post conditions of operations. Intuitively, a set of required pre, trigger, and post conditions operationalises a goal if satisfying the required conditions on operations guarantees that the goal is satisfied.

1. Preamble

2. Presentation
Icon, line style, text

User-definable attributes
[1:1] **Complete**: Boolean. Indicate whether the operationalisation is arguably sufficient (value "complete") or not arguably sufficient (value "undetermined").

[0:1] **AltName**: String ="". To name the corresponding alternative for further reference. In case a goal is operationalised into multiple alternative operationalisations this meta-attribute is mandatory.

Relations to other constructs
Belongs to 1..1 operation model.

1:1 [1:1], op_goal. **Goal** is operationalised through the **operation**.

0:n [1:1], op_operation. **Operation** operationalises the **goal**.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on
  - Goal
Operation

**Built on by**

**Instantiation level**
Instance and type level

**Classes of things**
1:1, “goalOwner” played by StakeholderThing.
   Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
   StakeholderThing is subclass of the BWW-HumanThing (specified in goal template).

1:1, “occurringEvent” played by ChangingThing.
   Operationalisation’s reqTrig includes a predicate Occurs on instances of that event.

1:1, “op_object” played by ComponentThing.
   Operation gets inputs as object attributes and produces outputs as object attributes.

**Properties (and relationships)**
1:1, “theGoal” played by ComplexLawProperty.
   Belongs to: 0:1 [1:1], goalOwner.
   Law: restricts the the possible values of the object attributes.
   Representing the goal which is held by a goal owner.

1:1, “terminalGoal” played by ComplexLawProperty.
   Sub-property: 1:1 [1:1], theAssignedGoal.
   State law: ∀g ∈ Goal, ∀a ∈ Assignment, a.assignedGoal = g
   ⇒ ¬∃gr ∈ G-refinement: gr.superGoal = g
   Only terminal goals can be assigned.

1:1, “operationalisedOperation” played by TransformationLaw.
   Sub-property: inputForOperation.
   Sub-property: outputForOperation.
   Belongs to: 0:n[0:n] occurringEvent.
   Transformation law: operation changes the states of the object.
   Describing the operationalised operation.
   Also represented by: operation.

0:n [0:n] “inputForOperation” played by BindingMutualProperty.
   Belongs to: op_object.
   Describing input for operation.
   Also represented by: Inputs.

0:n [0:n] “outputForOperation” played by BindingMutualProperty.
   Belongs to: op_object.
   Describing output for operation.
   Also represented by: Outputs.

1:1, “theOperationalisation” played by ComplexMutualProperty.
   Type: AND/OR relationship.
   Sub-property: 1:1[0:n] terminalGoal.
   Describing operationalisation. Operationalisation is a complex mutual property between goal owner, object and event.

1:1 [1:1], “attributeComplete” played by AnyRegularProperty.
   Sub-property: theOperationalisation.
   Represents operationalisation attribute complete.
KAOS Construct Analysis using the UEML Approach Template

0:1 [1:1], “attributeAltName” played by AnyRegularProperty.
   Sub-property: theOperationalisation.
   Represents operationalisation attribute altName.

1:n [1:1], “op_operation” played by ComplexProperty.

   Necessary condition that needs to be true when the operation is applied for the corresponding operationalised goal to be satisfied.

   Condition that needs to be established by the operation in its final state for the corresponding operationalised goal to be satisfied.

   Sub-property: 0:n [0:n] occurringEvent.
   Sufficient condition that requires the operation to be immediately applied for the corresponding operationalised goal to be satisfied.

Behavior
   Existence

Modality (permission, recommendation etc)
   Regular assertion

4. Open Issues
KAOS Construct Analysis using the UEML Approach Template
KAOS : Output

<table>
<thead>
<tr>
<th>Document type:</th>
<th>Working document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain/Task/Topic:</td>
<td>DEM / UEML / Approaches</td>
</tr>
<tr>
<td>Version:</td>
<td></td>
</tr>
<tr>
<td>Date:</td>
<td>2005.11.30</td>
</tr>
<tr>
<td>Status:</td>
<td>2nd iteration</td>
</tr>
<tr>
<td>Authors:</td>
<td>Raimundas Matulevičius</td>
</tr>
<tr>
<td>Distribution list:</td>
<td>DEM</td>
</tr>
</tbody>
</table>

An object is among the *outputs* of an operation if it is among the sorts making up the co-domain of the relation defined by the operation.

1. Preamble

Builds on

Built on by

Construct name

Outputs

Alternative construct names

Related, but distinct construct names

Related terms

*Inputs*: An object is among the *inputs* of an operation if it is among the sorts making up the domain of the relation defined by the operation.

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Operation model

2. Presentation

Builds on

Built on by

Icon, line style, text

![Diagram](image_url)
User-definable attributes

[0:1] `WhichAtt` : String = "". Indicate which attributes of the object are specifically monitored.

Relations to other constructs

- Belongs to 1..1 operation model.
- 0:n [1:1], Object. Object has output from an operation.
- 0:n [1:1], Operation. Operation outputs object attributes (changes its value).

Diagram layout conventions

3. Representation

Builds on

Built on by

Instantiation level

Type level

Classes of things

1:1 "outObject" played by ComponentThing.
- Represents object, which is output from operation.

0:n, "causingEvent" played by ChangingThing.

Properties (and relationships)

1:1, "theOutput" played by BindingMutualProperty.
- Belongs to: 0:n [1:1] object.
- Describing output. Output is a binding mutual property between the object which receives output from operation and the event which causes this operation.

1:n, "explicitObjAttribute" played by AnyProperty.
- Belongs to: outObject.
- Sub-property of: theOutput.
- Defines explicitly which attribute of the object is an output.

1:n, "implicitObjAttribute" played by AnyProperty.
- Belongs to: outObject.
- Sub-property of: theOutput.
- Does not define explicitly which attribute of the object is an output.

0:1, "attributeWhichAtt" played by AnyRegularProperty.
- Sub-property: theOutput.
- Representing the output attribute whichAtt.

0:n [1:1], "fromOperation" played by TransformationLaw.
- Belongs to: causingEvent.
- Sub-property: theOutput.
- Transformation law: operation produces output.
- Describing the operation which has output.
- Also represented by: operation.

Behaviour

Existence
Modality (permission, recommendation etc)

Regular assertion

4. Open Issues
KAOS : Performance

An operation is related to the agent that can initiate it through a performance link. Performance is an OR metarelationship linking agents to operations.

1. Preamble

Builds on

Built on by

Construct name

Performance

Alternative construct names

Performs

Related, but distinct construct names

Related terms

Language

KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type

Agent model

2. Presentation

Builds on

Built on by

Icon, line style, text
User-definable attributes

[0:1] Agent : Agent. Declare the agent.
[0:1] AltName : String = "". Name of alternative OR-assignments.

Relations to other constructs

• Belongs to 1..1 agent model.
• 1:1 [1:1], Agent. Agent performs operation.
• 1:n [1:1], Operation. Operation is performed by agent.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on
Agent

Built on by

Instantiation level
Type level

Classes of things

1:1, "goalOwner" played by StakeholderThing.
Describing goal owner which holds the goal. Class StakeholderThing has a characteristic holds goal.
StakeholderThing is subclass of the BWW-HumanThing.

1:1 “responsibleAgent” played by ActiveComponentThing.
Represents the responsible agent.

Properties (and relationships)

1:1, “theGoal” played by ComplexLawProperty.
Belongs to: 0:1 [1:1], goalOwner.
Law: restricts the the possible values of the object attributes.
Representing the goal which is held by a goal owner.

1:1, “terminalGoal” played by StateLaw.
Belongs to: 0:1 [1:1], goalOwner.
Sub-property: 1:1 [1:1], theGoal.
State law: \( \forall g \in \text{Goal}, \forall a \in \text{Assignment}, a.\text{assignedGoal} = g \)
\( \Rightarrow \neg \exists gr \in \text{G-refinement}: \text{gr.superGoal} = g \)
Only terminal goals can by assigned.

1:1, “thePerformance” played by ComplexMutualProperty.
Type: OR-relationship.

Diagram:

```
  Agent name
     ^
     |
     |
     |
     |
     |
     Operation name

  Performance
```
Belongs to: 1:n [1:1] responsibleAgent.
Representing the performance. Performance is a complex mutual property of the responsible agent and the goal owner. Agent performs the operation in order to satisfy the goal.

0:1, “attributeAgent” played by AnyRegularProperty.
Sub-property: thePerformance.
Representing the performance attribute agent.

0:1, “attributeAltName” played by AnyRegularProperty.
Sub-property: thePerformance.
Represents performance attribute altName.

1:1 [1:1], “performedOperation” played by TransformationLaw.
Sub-property: thePerformance.
Transformation law: agent performs operation to satisfy the goal. Describing the operation which is performed by agent.
Also represented by: operation.

1:n [1:1], “operationalisation” played by ComplexMutualProperty.
Sub-property: performedOperation.
Sub-property: terminalGoal.
Describing the operationalisation relationship.
Also represented by: operationalisation.

Behaviour
Existence

Modality (permission, recommendation etc)
Regular assertion

4. Open Issues
KAOS : Requirement

A requirement is a goal assigned to an agent in the software-to-be.

1. Preamble

Builds on
   Goal

Built on by

Construct name
   Requirement

Alternative construct names

Related, but distinct construct names
   Terminal goal: goal which has no G-requirement.

Related terms
   Expectation (assumption): a goal assigned to an agent in the environment.
   Softgoal: a goal that cannot be said to be satisfied in a clearcut sense.
   Comment: can a requirement be a softgoal?
      Maintain goal: a goal requiring that some property always holds.
      Avoid goal: a goal requiring that some property never holds.
      Achieve goal: a goal requiring that some property eventually hold.
      Cease goal: a goal requiring that some property eventually stops to hold.

Language
   KAOS,
   http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type
   Goal model
2. Presentation

Builds on

Built on by

Icon, line style, text

User-definable attributes
Requirement inherits all the attributes of the goal.

Relations to other constructs
- Belongs 1:1 to goal model.
- 1:n [1:n], responsible: software agent. Requirement is assigned through responsibility relationship to a software agent.

Diagram layout conventions

Other usage conventions

3. Representation

Builds on
Comment: Requirement is a goal and has most of the goal classes and properties. But requirement is also a terminal goal, so it has no G-refinement.

Built on by

Instantiation level

Instance level

Classes of things
1:1, “softwareAgent” played by ActiveComponentThing.
Describing the agent a requirement is assigned.

Properties (and relationships)
1:1, “theRequirement” played by ComplexStateLaw.
   Belongs to: 0:1 [0:n] softwareAgent.
   Belongs to: 0:1 [1:1] goalOwner.
   State law: Is restricted by the assignement relationship. A requirement himself restricts state of the concerned object.
   Representing the requirement. Requirement as a goal, has a goal owner.
1:1, “isTerminalGoal” played by StateLaw.
   Sub-property: 1:1 [1:1], theRequirement.
   State law: ∀g ∈ Goal, ∀a ∈ Assignment, a.assignedGoal = g
⇒ ¬∃ gr ∈ G-refinement: gr.superGoal = g

*Requirement is a terminal goal* which means that an *requirement* can not have G-refinement.

**Behaviour**

Existence

**Modality (permission, recommendation etc)**

*Intention* of a goal owner;

*Obligation* of a *software agent*.

4. Open Issues
A *softgoal* is a goal that cannot be said to be satisfied in a clearcut sense. It prescribes classes of preferred behaviour.

1. **Preamble**

   **Builds on**
   - Goal

   **Built on by**

   **Construct name**
   - Softgoal

   **Alternative construct names**
   - Goal
   - Preferred behaviour

   **Related, but distinct construct names**

   **Related terms**
   - *Requirement*: a goal assigned to an agent in the software to be.
   - *Assumption*: a goal assigned to an agent in the environment.

   **Language**
   - KAOS,

   **Diagram type**
   - Goal model

2. **Presentation**

   **Builds on**

   **Built on by**
Icon, line style, text

User-definable attributes
[0:1] Type : set_of [Minimize, Maximize, Reduce, Increase, Improve]. Describes type of the softgoal.

Relations to other constructs

Diagram layout conventions

Other usage conventions

3. Semantics

Builds on
Comment: Softgoals can be And/Or refined like any other KAOS goals, conflicts between softgoals goals can also be captured. An important research issue concerns the precise definition of optimization goals, reasoning techniques about softgoals, and the role of such goals in selecting among alternative goal refinements.

Built on by

Instantiation level

Classes of things

Properties (and relationships)

1:1, “theSoftGoal” played by ComplexLawProperty.
Representing the softgoal.

1:1, “notSatisfiedClearly” played by AnyProperty.
The softgoal has property (rather feature) not to be satisfied in a clearcut sense.

0:1 [1:1], “attributeType” played by AnyRegularProperty.
Sub-property: theSoftGoal.
Represents goal attribute type.

Behaviour
Existence

Modality (permission, recommendation etc)

Intention of a goal owner;

4. Open Issues

Comment: Further investigation about all softgoal (as goal) features is needed!
KAOS : Software agent

Software agent is an agent in the system-to-be.

1. Preamble

Builds on
Agent

Built on by

Construct name
Software agent

Alternative construct names
Agent in the system-to-be

Related, but distinct construct names

Related terms
- Agents : active objects capable of performing operations.
- Environment agent : e.g., pre-existing software component, sensor, actuator, human, organisational unit, etc.

Language
KAOS,
http://www2.info.ucl.ac.be/research/projects/AVL/ReqEng.html

Diagram type
Agent model

2. Presentation

Builds on

Built on by

Icon, line style, text

AgentName

Software agent
User-definable attributes

Relations to other constructs

Diagram layout conventions

Other usage conventions

3. Representation

Builds on

Built on by

Instantiation level
Type level

Classes of things
1:1, “theSoftwareAgent” played by ComponentSoftwareThing.
Representing the software agent. Software agents are agents and inherits all agent attributes and properties.

Properties (and relationships)

Behaviour
Existence

Modality (permission, recommendation etc)
Regular assertion

4. Open Issues
KAOS Construct Analysis using the UEML Approach Template