Analysis of KAOS Meta-model

Matulevicius, Raimundas; Heymans, Patrick

Publication date:
2005

Namur University
Computer Science Department

Analysis of KAOS Meta-model

(Technical report)

Raimundas Matulevičius
and Patrick Heymans

Namur, Belgium 2005

The KAOS approach consists of a modelling language, a method, and a software environment. In this paper, we will consider only the KAOS modelling language – simply called KAOS, from now on. A KAOS model includes a goal model, an object model, an agent model and an operation model. Each of them has a graphical and a textual syntax. Selected constructs can be further defined using the KAOS real-time temporal logic facilitating rigorous reasoning.

The scope of this work is to determine the precise semantics of the KAOS constructs. The KAOS application consists of four basic models [1]: goal model, object model, agent model, and operation model. We are using two main sources of the KAOS meta-model in our study. A part of the meta-model is exposed by Letier in [2] through structures and "meta-constraints" described in conventional mathematics but intertwined with other topics. In [1] Lamsweerde focuses only on the meta-model, but uses undefined notations and non-standard constructions to visualise it. However, this work also omits some multiplicities, specialisation-related constraints and abstract classes. Furthermore, integrity constraints are only given partially and informally.

In order to facilitate the analysis, we materialised our understanding of KAOS in a UML 2.0 class diagram given in Fig. 1. In Table 1 we also list a number of limitations of the meta-models defined in [1] and [2]. Furthermore Table 2 defines a list of integrity constraints for our meta-model.

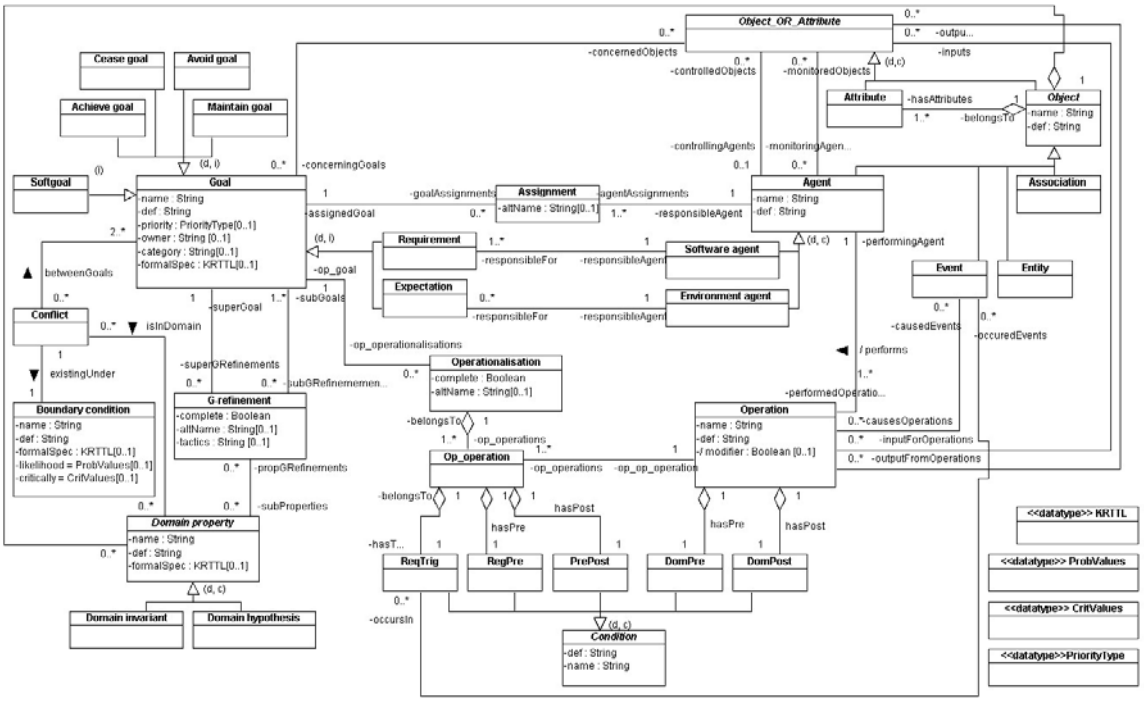


Fig. 1. Our meta-model for KAOS goal model

We show some KAOS concrete syntax in Fig. 2 in the example on the London Ambulance Service system adapted from [2]. The focus of our analysis is the *goal model*; however, agent, object and operation models are not excluded completely, as all models are interrelated.

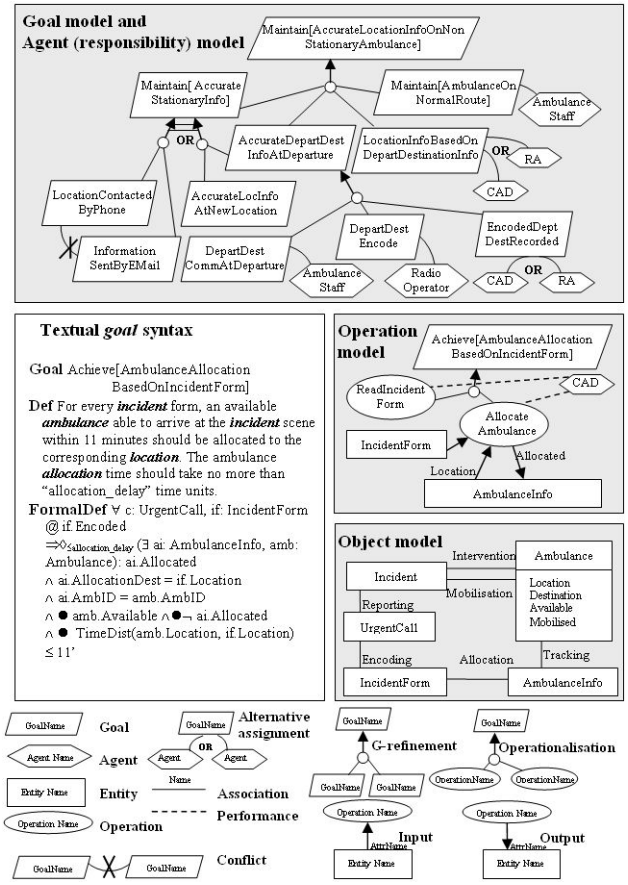


Fig. 2. Fragment of the KAOS model for the London Ambulance Service system (adapted from [2])

A ***goal*** is a prescriptive assertion that captures an objective which the system-to-be should meet [1, 2]. Goals can be classified according to one of four *patterns*: *maintain*, *avoid*, *achieve* and *cease* goals:

- ***Maintain*** goals require that some property always holds.
- ***Avoid*** goals require that some property never holds.
- ***Achieve*** goals require that some property eventually holds.
- ***Cease*** goals requires that some property eventually stops to hold.

An optional attribute *category* provides a further classification of goals according to their services provided to the agent (functional goals) or some quality of services (non-functional goals). Category could define *satisfaction* goals (concerning with satisfying agent wishes), *safety* goals (concerning with avoiding hazardous states), *accuracy* goals (concerning the accuracy of beliefs of an agent about its environment) and others.

A goal can be refined through *G-refinement*, which relates it to a set of *subgoals* whose conjunction, possibly together with *domain properties*, contributes to the satisfaction of the goal.

Domain property is a descriptive assertion about *object* in the environment which holds independently of the system-to-be [1, 2]. Domain property that is naturally true about objects can be declared as *domain invariants*. A *domain hypothesis* is a property about some domain object supposed to hold and used when arguing about the sufficient completeness of goal refinement.

G-refinement links are AND/OR relationships. G-refinement presents one alternative (alternative indicated by *altName* optional attribute) set of subgoals whose conjunction, possibly together with domain properties, contributes to the satisfaction of the parent goal. A goal can have alternative G-refinements which result in different software designs.

A set of goals is ***conflicting*** if these goals cannot be achieved together. This means that under some ***boundary condition*** these goals become logically inconsistent in the considered domain. Conflict

management is modelled by a tenary conflict meta-relationship between goal, domain property and boundary condition.

Softgoal is a goal that does not have a clear-cut criterion for its satisfaction. Instead of goal satisfaction, goal *satisficing* is introduced to express that subgoals are expected to contribute to the satisfaction of the softgoal within acceptable limits rather than absolute. Softgoals can be refined like any other KAOS goals, *conflicts* between softgoals can also be captured [2].

The *object model* is not analysed in detail, but it is restricted to relationships with the goal model. An ***object*** is a thing of interest in the system being modelled whose instances can be distinctly identified and may evolve from state to state [1, 2]. At the application level objects may be organised in inheritance and aggregation hierarchies [1]. Objects are not necessary disjoint. An object at the instance level could simultaneously be an instance of the two different objects [2]. An object is an *entity*, *association* (*relationship* in [2]), *event* or *agent* depending on whether the object is autonomous, subordinate, instantaneous or active [1]. Goals *concern* objects (see *Def* in textual goal syntax in Fig. 2).

The *agent model* is not analysed in detail, but it is restricted to relationships with the goal model. An ***agent*** is an active object which plays a specific role towards goal achievement by controlling specific object behaviour [1, 2]. ***Assignment*** relationship is defined as *possible* assignment of a goal to an agent. *Responsibility* link defines an *actual assignment* of a goal to an agent. A goal effectively assigned to a ***software agent*** is called a ***requirement***. A goal effectively assigned to an ***environment agent*** is called an ***expectation*** (*assumption* in [2]). Like G-refinement, assignment has the optional *AltName* attribute which indicates agent responsible for goal satisfaction. In case of multiple alternatives of assignments this attribute is mandatory.

An agent ***monitors*** or ***controls*** an *object* if the state of the object is directly observable or controllable by agent. An optional *WhichAtt* meta-attribute is attached to the monitoring and control meta-relationships to allow explicitly indicate which attributes of the object are monitored or controlled [1]. The object monitored by an agent is observable by that agent as well (see IC #9). The object controlled by an agent is also modified by that agent (see IC#10).

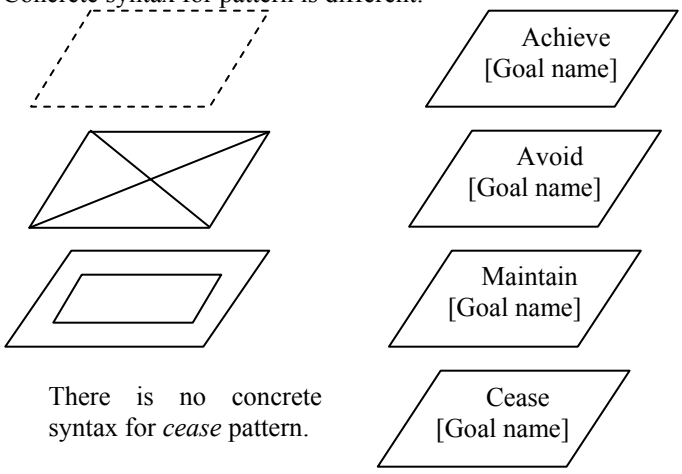
The *operation model* is not analysed here in detail, but it is restricted to relationships with the goal model. An ***operation*** is an ***input-output*** relation over objects; operation application defines state transition [1, 2]. Operations are characterised by pre-, post-, and trigger conditions. A distinction is made between *domain* pre/post conditions, which capture the elementary state transitions defined by operation application in the domain, and *required* pre/trigger/post conditions (see *operationalisation* in Fig.2), which capture additional strengthening to ensure that the goals are met.

The meaning of the conditions is defined in [2]. *DomPre* characterises the states before any application of the operation. *DomPost* defines a relation between states before and after applications of the operation. *ReqPre* defines those states in which the operation is allowed to be applied. *ReqTrig* defines those states in which the operation is obliged to be immediately applied provided the domain precondition is true. *ReqPost* defines additional conditions that applications of the operation must satisfy. The operation has an optional derived attribute *modifier* in order to indicate whether the operation is an object of *Modifier* (equals 1) or *Observer* (equals 0) – see also IC #13, IC #14 and IC #15. Operation might be caused by ***events***.

The ***operationalisation*** meta-relationship is an AND/OR relationship between goals and required pre, trigger, and post conditions. 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 [2].

If a goal is operationalised and has a responsible agent, and the latter ***performs*** the operations (see IC #11). All the goals operationalised by the same operation must have the same (actual) responsible agent (see example in Fig. 1 and IC #12).

Table 1. Limitations of KAOS meta-models [1, 2] and our assumptions

	Limitations		Our assumptions
	Meta-model presented in [1]	**Meta-model presented in [2]**	
Goal patterns and categories	It is not identified whether the pattern classification is complete. Are there more patterns than *maintain*, *avoid*, *achieve*, and *cease*?		Yes
	Concrete syntax for pattern is different:  There is no concrete syntax for *cease* pattern.		We prefer notations used in [2]
	It does not specified whether the goal can be classified to more than one category.		Yes
		Relationship between pattern and category is specified. But it is not defined if this relationship is *complete/incomplete*, and *overlapping/disjoint*.	
Domain properties	It is not specified whether the domain hypothesis and domain invariants are overlapping or not?		Disjoint and complete
		Only domain invariant is defined. Further term '*domain theory*' is used without explanation.	
G-refinement	What is cardinality between goals, G-refinement and domain properties?		A goal could be refined through one-to-many subgoals, while there might be zero-to-many domain properties in the G-refinement.
	Can a goal be a subgoal in two G-refinements of the same supergoal (see in Fig.3.a)?		Yes (*AccurateDepartDestInfo-AtDepartment* in Fig. 1)
	Can two goals have subgoals in common (see Fig. 3.b)?		Yes
	Can there be loops in G-refinement?		No (see IC#1)
	G-refinement can be *complete* or *underdetermined*. Is there a third option (e.g. *unknown*)?		No

Softgoal	Softgoals are latter edition of the KAOS meta-model. It is not defined if the softgoals can have patterns, if the softgoals can be requirements or expectations, whether they can have concerns or be operationalised (and so on).		Left for future research
	Are softgoal, patterns and requirement/expectation subtypes disjoint wrt one another?		No
Object	Association	Relationship	Association
	Because of the complexity of the object model it is expected problems and limitation in the future analysis of this model.		
	Goal formulation in *Def* attribute refers to the objects and their attributes.	The objects model declares the vocabulary to be used in the definition of goals; these definitions bound the vocabulary to be declared in the object model	Assume [1]. [2] might be supplementary.
	The multiplicity of a goal to concerns object is 1..*. But the textual explanation (see above) confuses when defining the goal concern over object attributes.		Multiplicity is defined 0..* meaning that a goal can concern an object or its attribute through the abstract class *Object_OR_Attribute* (see IC #3)
	"*concern*" consistency with the G-refinement is not taken in consideration.		
Assignment/ responsibility	Is a *terminal goal* (one that is not the supergoal in a G-refinement) necessarily a requirement or an expectation?		No (see IC #4)
	Do assignment and responsiblity relate goals to agent *classes*, *instances* or *both*? (we assume only *classes* - simplification)		Classes (simplification, due to very loosely defined KAOS Object Model)
	Are *responsibility* and *assignment* the same?		No (see IC #5 and IC #6)
	Is the difference between *requirement* and *expectation* based on assignment or responsibility?		On responsibility (see IC #7 and IC #8)
	Expectation	Assumption	Expectation
		A *condition* is sometimes called *requirement*	Improper use of *requirement*.
	Multiplicity defines that each agent should be responsible for the goal.		We assent to this constraint. But we point out that agents are organised into hierarchies (as *objects*) and some agents will no be assigned to any goal.
Monitors/ controls	It is not identified whether the object or attribute is monitored or controlled. It is not clear what is declared by the "object state".		Monitoring and controlling relationships to the *Object_OR_Attribute* abstract class.
Conditions	All the conditions must be defined informally, but precisely.	Conditions are defined using KAOS temporal logic.	Both

Operationalisation	Conditions reqPre, reqTrig and reqPost are defined both for the operationalisation and operationalisation.	Conditions reqPre, reqTrig and reqPost are defined only for the operationalisation	Definition of a class "*Op_operation*" where the required conditions are defined.
	It is not clarified which goals should be operationalised.		Only the goals which have a responsible agent could be operationalised.
	It is not defined whether all the requirements and expectations have to be operationalised in a model.		The transition of the constructs while performing modelling activities could change during intermediate phases, thus, the goals can be not operationalised. A at the *final model* all the requirements and expectations must be operationalised *or operationalised at the feasible level* in order to ensure the satisfaction of the goals.
	Operationalisation can be *complete* or *underdetermined*. Is there a third option (e.g. unknown)?		No (see IC #13)
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." By how many events an operation can be *caused*?		0..* (see IC #17)

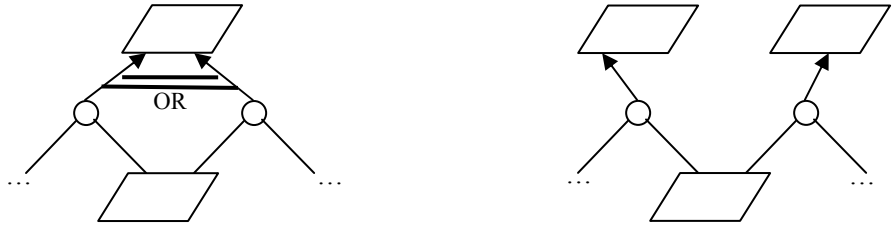


Fig. 3. Subgoal role in G-refinement

Table 2. Integrity constraints for our meta-model.

IC No	Textual explanation	OCL constraints
IC #1	G-refinement admits no loops.	***context*** *Goal* ***inv :*** *Goal->* ***forall****(g2 | not (self.ancestor->includes(g2) and g2.ancestor->includes(self)))*
IC #2	Two alternative G-refinements could not have the same name.	***context*** *G-refinement* ***inv :*** *G-refinement->* ***forall****(gr2 | self.altName=gr2.altName* ***and*** *self.supergoal=gr2.supergoal* ***implies*** *self=gr2)*

IC #3	For each object there exists a goal which concerns object or its attribute.	***context:*** *Goal* ***inv:*** *Object ->* ***forall****(o | self.concernedObject->includes(o)* ***or*** *self.concernedObject -> includes(o.hasAttribute))*
IC #4	Terminal goal does not have G-refinement	***context:*** *Assignment* ***inv:*** *G-refinement ->* ***implies not exists*** *(gr | self.assignedGoal = gr.superGoal)*
IC #5	A goal effectively assigned to a software agent is called requirement.	***context*** *Requirement* ***inv:*** *SoftwareAgent ->* ***exists*** *(a | self.goalAssignments = a.agentAssignments)*
IC #6	A goal effectively assigned to an *environment agent* is called an *expectation*.	***context*** *Expectation* ***inv:*** *EnvironmentAgent ->* ***exists*** *(a | self.goalAssignments = a.agentAssignments)*
IC #7	The agent deemed responsible (actual responsibility) for a requirement is one to which the goal was assigned (possible responsibility).	***context*** *Requirement* ***inv :*** *Assignement->* ***exists****(as | as.assignedGoal=self* ***and*** *as.responsibleAgent-> includes(self.responsibleAgent))*
IC #8	The agent deemed responsible (actual responsibility) for an expectation is one to which the goal was assigned (possible responsibility).	***context*** *Expectation* ***inv :*** *Assignement->* ***exists****(as | as.assignedGoal=self* ***and*** *as.responsibleAgent-> includes(self.responsibleAgent))*
IC #9	The object monitored by an agent is observable by that agent as well.	***context:*** *Agent :: monitoredObjects : Set(Object_OR_Attribute)* ***inv:*** *Operation ->* ***exists*** *(o | self.performedOperation = o* ***and*** *o.inputs -> □sset()))* ***and not exists*** *(o2 | self.performedOperation = o2* ***and*** *o2.inputs -> □sset())*

IC #10	The object controlled by an agent is also modified by that agent.	***context:*** *Agent :: controlledObjects : Set(Object_OR_Attribute)* ***inv:*** *Operation ->* ***exists*** *(o | self.performedOperation = o* ***and*** *o.outputs-> isset()))*
IC #11	The agent that performs an operation is the (actual) responsible agent for the goal (requirement or expectation) that the operation operationalises.	***context*** *Agent ::performs : Set(Operations)* ***derive :*** *self.responsibleFor.op_operationalisations.op_operation.op_op_operation ->asSet()*
IC #12	All the goals operationalised by the same operation must have the same (actual) responsible agent	***context*** *Goal* ***inv :*** *Goal->forall(g2 | self.operationalisations->includes(op)* ***and*** *op.op_operation->includes(op_op)* ***and*** *op_op.operation=o* ***and*** *g2.operationalisations->includes(op2)* ***and*** *op2.op_operation->includes(op_op2)* ***and*** *op_op2.operation=o* ***implies*** *self.responsibleAgent=g2.responsibleAgent)*
IC #13	If operationalisation is complete for the goal then all objects and attributes concerned by this object are input or aoutput for at least one operation.	***context*** *Operationalisation* ***inv:*** *Object_OR_Attribute (oa | self.complete = TRUE* ***and*** *oa -> includes(self.op_goal.concernedObjects)* ***implies*** ***exists*** *(oa.inputForOperations.op_operations.belongsTo->includes(self))* ***or*** *(oa.outputForOperations.op_operations.belongsTo->includes(self)))*
IC #14	An operation is odifier if it has an output.	***context*** *operation :: modified=TRUE* ***derive:*** *Object_OR_Attribute ->* ***exists*** *(oa | oa.inputForOperations -> includes(self))*
IC #15	[*IC #15 is defined to avoid observers without having anything to observe (operations without input). IC#16 is redundant to IC #15.*]	***context*** *operation* ***derive:*** *Object_OR_Attribute ->* ***exists*** *(oa | oa.outputFromOperations -> includes(self)* ***or*** *oa.inputForOperations -> includes(self))*
IC #16	An operation is observer if it has an input.	***context*** *operation :: modified = FALSE* ***derive:*** *Object_OR_Attribute ->* ***exists*** *(oa | oa. inputForOperations -> includes(self))* ***and not exists*** *(oa2 | oa2.outputFromOperation -> includes(self))*
IC #17	One operation might be caused by events.	***context*** *Event :: causesOperations : Set(Operation)* ***inv:*** *Event -> self.occursIn.belongsTo.op_op_operation->asSet()*

References

[1] van Lamsweerde, A. The KAOS Metamodel – Ten Years After. University of Louvain, Internal report, 2003.
[2] Letier E., Reasoning about Agents in Goal-Oriented Requirements Engineering. PhD theses. Universite catholique de Louvain, 2001, pp 295.