A Framework to Evaluate Complexity and Completeness of KAOS Goal Models

Patrícia Espada, Miguel Goulão, and João Araújo

CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Lisbon, Portugal
titiespada@gmail.com,
{mgoul,joao.araujo}@fct.unl.pt

Abstract. Goal-Oriented Requirements Engineering (GORE) approaches have been developed to facilitate the requirements engineers work by, for example, providing abstraction mechanisms to help eliciting and modeling requirements. One of the well-established GORE approaches is KAOS. Nevertheless, in large-scale systems building KAOS models may result in incomplete and/or complex goal models, which are difficult to understand and change. This may lead to an increase in costs of product development and evolution. Thus, for large-scale systems, the effective management of complexity and completeness of goal models is vital. In this paper, we propose a metrics framework for supporting the quantitative assessment of complexity and completeness of KAOS goal models. Those metrics are formally specified, implemented and incorporated in a KAOS modeling tool. We validate the metrics with a set of real-world case studies and discuss the identified recurring modeling practices.

Keywords: Goal-Oriented Requirements Engineering, Requirements Metrics, Model Complexity, Model Completeness.

1 Introduction

Goal-Oriented Requirements Engineering (GORE) is considered an established paradigm in requirements engineering to handle elicitation, specification, analysis, negotiation and evolution of requirements by using goals [1]. KAOS [2], i* framework [3], GBRAM [4] and GRL [5] are among the most representative GORE approaches. In this paper, our focus is on the KAOS approach.

GORE approaches were developed to support the development of large-scale systems by providing different models, where the goal model is naturally the central one. Eliciting requirements for such large-scale models is typically performed in a step-wise manner. The higher-level goals are decomposed into less abstract goals. In this refinement process, it is useful to have a measure of completeness, which can help practitioners realize how close they are to achieving model completion.

Another challenge is that while designing such systems with the help of GORE approaches it is common to reach a point where the models are so complex that their analysis becomes difficult. Part of this complexity is intrinsic to the system. However, complexity can also be accidental, i.e., it can result from the way the approach is used to build the models [6]. From this point of view, one should minimize the accidental complexity of models as a way to improve their quality.

In [7] we made an initial proposal to evaluate completeness and complexity of KAOS goal models. By applying manually these metrics to two versions of an example, one being a refinement of the other, we illustrated how completeness and complexity metrics could be used by requirements engineers to better manage their models. Requirements engineers need models that are relatively easy to understand, to facilitate requirements evolution. Tool support should be used to evaluate such models, giving useful feedback to the user while building them. This would help them (1) to know the extent to which a model is close to being complete – this can be very hard to acknowledge in large models; (ii) to assess the complexity of models and identifying model refactoring opportunities by locating, for example, models that have a very deep goal hierarchy or agents with too many responsibilities; and (iii) to prevent unanticipated extra costs in the development phase, as a result by better managing the completeness and complexity of the models.

In this paper, we propose a tool supported approach to assist requirements engineers in the evaluation of the completeness and complexity of KAOS goal models, in an incremental way, while building those models. The developer can measure the current status of his model and take on corrective actions, if necessary, during model construction. The tool support is based on the integration of a KAOS editor with a KAOS metrics suite and is mostly targeted to the requirements elicitation process, although it can also support *post-mortem* analysis from which lessons can be learned for future projects. We extend our previous work to include new metrics and formally define all metrics using Object Constraint Language (OCL) [8]. We then validate the metrics set and their implementation by extending an existing tool for editing KAOS goal models (modularKAOS) [9], which was developed in an Eclipse platform by using Model-Driven Development (MDD) techniques.

This paper is organized as follows. Section 2 describes background information on KAOS. Section 3 describes the metrics set (defined using the GQM approach) and a concrete example of its application to a real-world model. Section 4 reports the evaluation process, including a presentation of the case studies used, the results obtained by applying the metrics on those case studies, and a discussion on the results. Section 5 discusses the related work. Section 6 draws some conclusions and points out directions for future work.

2 Background

KAOS is a GORE framework whose emphasis is on semi-formal and formal reasoning about behavioral goals to derive goal refinements, operationalizations, conflict management and risk analysis [2]. In KAOS, goals can be refined into subgoals through and/or decompositions. Goals can also be refined into requirements (i.e., a goal whose responsibility is assigned to a software agent), or expectations (i.e., a goal whose responsibility is assigned to an environment agent). KAOS also introduces the concept of obstacle as a situation that prevents the achievement of a goal [10]. The resolution to the obstacle is expressed in the form of a goal that can also be refined.

The main steps for building KAOS specifications from high level goals are [2]: **(i) goals development** – identification of goals and their refinement; **(ii) objects identification** – objects identification in the formulation of the goal, definition of the links among them, and description of the domain properties; **(iii) operations identification** – identification of object state transitions to the goal; **(iv) goals operationalization** – specification of operations to satisfy all goals; **(v) responsibilities assignment** – mapping of agents to leaf goals and operations assignment to agents.

We use the modularKAOS approach and tool [9], which includes a Domain-Specific Language (DSL), implemented using MDD techniques, for building well-formed KAOS models. This DSL was implemented based on the metamodel defined in [11], using Ecore [12]. Our metrics suite is implemented and integrated in this tool. The modularKAOS metamodel can be found in [9].

3 The KAOS Goal Model Evaluation Metrics

The purpose of this study is to evaluate the completeness and complexity of KAOS goal models from the perspective of requirements engineers in the context of GORE. Metrics can be valuable to analyze these properties. We propose a metrics-based analysis framework for KAOS models, using the Goal-Question-Metric (GQM) approach [13]. Table 1 summarizes our GQM-based proposal for a set of metrics that will allow satisfying the goals of completeness and complexity evaluation. For each goal, the first column presents questions that will allow evaluating whether the corresponding goals are being achieved. The second column shows the respective metrics.

Table 1. GQM for KAOS goal models evaluation

Goal: Completeness	
Question	**Metric**
Q1. How close are we to completing the assignment of all goal responsibilities to agents?	**PLGWA**. **P**ercentage of **L**eaf **G**oals **W**ith an **A**gent.
Q2. How detailed is the goal model with respect to objects?	**PLGWO**. **P**ercentage of **L**eaf **G**oals **W**ith an **O**bject.
Q3. How close are we to complete the resolution of all the goal obstacles?	**PLOWS**. **P**ercentage of **L**eaf **O**bstacles **W**ith a re**S**olution.
Q4. How detailed is the goal model with respect to operations?	**PLGWOp**. **P**ercentage of **L**eaf **G**oals **W**ith an **Op**eration.
Q5. How well supported are the operations in the goal model?	**POpWA**. **P**ercentage of **Op**erations **W**ith an **A**gent.
Goal: Complexity	
Q6. Does an agent have too much responsibility in the model?	**ANLG**. **N**umber of **L**eaf **G**oals per **A**gent.
Q7. Does a leaf goal have too many/few objects?	**GNO**. **N**umber of **O**bjects per **G**oal.
Q8. How difficult is it to understand a model, with respect to the number of refinement levels?	**MD**. **M**odel **D**epth.
Q9. How complex is a model, with respect to its goal refinements?	**RNSG**. **R**oot **N**umber of **S**ub-**G**oals.

To achieve the completeness goal for KAOS goal models, we formulated five different questions. Those address: (Q1) attributing responsibilities of goals to agents, (Q2) associating objects to goals, (Q3) providing resolutions to obstacles, (Q4) associating operations to goals, and (Q5) associating operations to agents.

Concerning the complexity goal, we defined four questions that address (Q6) the amount of responsibility supported by an agent in a model, (Q7) the number of objects associated to a leaf goal, (Q8) the understandability of the goal model with respect to the refinement levels (i.e., depth of goal hierarchy), and (Q9) the number of goal refinements.

3.1 Metrics Definition

Table 2 presents the set of questions related to the evaluation of the completeness goal. Table 3 defines the set of questions concerning the complexity goal. For each question we present an informal definition of the metric specified to answer it, and its formal definition using OCL upon the modularKAOS metamodel. Whenever needed, we include the formal pre-conditions for the metrics computation. These define when it makes sense to compute a metric (e.g., if we measure the percentage of leaf goals in the model which have an agent and there are no leaf goals, it makes no sense to compute the metric). We also provide an informal definition for the auxiliary metrics used in each definition[1]. The included comments further explain the main metric for each question. Last, but the least, we provide practical recommendations for interpreting each of the proposed metrics, thus helping practitioners to monitor the completeness and complexity of their models.

For the sake of completeness of the goal model, each leaf goal must be assigned to an agent [1]. The PLGWA metric addresses this issue. This metric also covers the case where a parent goal is assigned to an agent. In such cases, the sub-goals are also assigned to that agent. Associating objects with leaf goals is optional, but contributes to model completeness with information that is used in later stages of system development. The PLGWO metric captures this perspective of completeness. Providing a resolution to obstacles contributes to the completeness of a KAOS goal model. The PLOWS metric measures the obstacle resolution coverage. Operations represent well determined solutions that fulfill the goals. A model does not necessarily need to have all its goals operationalized. One may deliberately decide to postpone the operationalization of a goal to a later stage. The PLGWOp metric can be used to assess the extent to which the model goals are realized. Finally, KAOS operations must be performed by agents. The POpWA metric evaluates the percentage of operations with an assigned agent.

Table 2. Metrics to satisfy the completeness goal – Q1 through Q5

Q1 - How close are we to completing the assignment of all goal responsibilities to agents?	
Name	**PLGWA – P**ercentage of **L**eaf **G**oals **W**ith an **A**gent
Informal definition	Percentage of leaf goals that have an associated agent in the model.

[1] All metrics are available at `http://ctp.di.fct.unl.pt/~mgoul/papers/2013/CAISE2013KAOSMetrics.pdf`

Table 2. (*continued*)

Formal definition	**context KAOS** **def: PLGWA(): Real** = self.NLGWA() / self.NLG()
Pre-condition	context KAOS::PLGWA() pre: self.NLG() > 0
Comments	If there are no leaf goals the result is undefined. This requires: **NLG – N**umber of **L**eaf **G**oals **NLGWA**– **N**umber of **L**eaf **G**oals **W**ith an **A**gent
Recommendation	In a complete model, all leaf goals should be assigned to an agent.
Q2 - How detailed is the goal model with respect to objects?	
Name	**PLGWO – P**ercentage of **L**eaf **G**oals **W**ith an **O**bject
Informal definition	Percentage of leaf goals that have an associated object in the model.
Formal definition	**context KAOS** **def: PLGWO(): Real** = self.NLGWO() / self.NLG()
Pre-condition	context KAOS::PLGWO() pre: self.NLG() > 0
Comments	If there are no leaf goals the result is undefined. This requires: **NLGWO** – **N**umber of **L**eaf **G**oals **W**ith an **O**bject **NLG – N**umber of **L**eaf **G**oals
Recommendation	In a complete model, goals should have associated objects.
Q3 - How close are we to complete the resolution of all the goal obstacles?	
Name	**PLOWS – P**ercentage of **L**eaf **O**bstacles **W**ith a goal re**S**olution
Informal definition	Percentage of leaf obstacles that have an associated goal resolution in the model.
Formal definition	**context KAOS** **def: PLOWS(): Real** = self.NLOWS() / self.NLO()
Pre-condition	context KAOS::PLOWS() pre: self.NLO() > 0
Comments	If there are no leaf obstacles the result is undefined. This requires: **NLO – N**umber of **L**eaf **O**bstacles **NLOWS – N**umber of **L**eaf **O**bstacles **W**ith a re**S**olution
Recommendation	Resolutions should be assigned to obstacles to contribute to the resilience of the system to exceptional situations.
Q4 - How detailed is the goal model with respect to operations?	
Name	**PLGWOp – P**ercentage of **L**eaf **G**oals **W**ith an **Op**eration
Informal definition	Percentage of leaf goals that have at least one associated operation in the model.
Formal definition	**context KAOS** **def: PLGWOp(): Real** = self.NLGWOp() / self.NLG()
Pre-condition	context KAOS::PLGWOp() pre: self.NLG() > 0
Comments	If there are no leaf goals, this metric is undefined. This requires: **NLGWOp – N**umber of **L**eaf **G**oals **W**ith an **Op**eration **NLG – N**umber of **L**eaf **G**oals
Recommendation	Operations should be assigned to leaf goals, to get closer to completing the specification of the desired functionalities.
Q5 - How well supported are the operations in the goal model?	
Name	**POpWA – P**ercentage of **Op**erations **W**ith an **A**gent
Informal definition	Percentage of operations that have an associated agent in the model.
Formal definition	**context KAOS** **def: POpWA(): Real** = self.NOpWA() / self.NOp()
Pre-condition	context KAOS::POpWA() pre: self.NOp() > 0
Comments	If there are no operations, this metric is undefined. This requires: **NOp – N**umber of **Op**erations **NOpWA– N**umber of **Op**erations **W**ith an **A**gent
Recommendation	All operations must be performed by agents, so assigning agents to operations is a necessary step to model completion.

Table 3 presents the metrics to satisfy the complexity goal. The ANLG metric counts the number of leaf goals assigned to an agent, including the inherited ones. It measures the complexity of that agent in terms of the amount of its responsibilities. The GNO metric addresses the complexity related to the number of objects associated to a leaf goal. It is essentially a size metric of the model that can be used to evaluate if a goal has an associated complex behavior by manipulating a certain amount of objects. A deeper model hierarchy makes it harder to understand the rationale for a leaf goal or an obstacle. The MD metric measures the height of the goal graph. Finally, the RNSG metric represents the number of sub-goals resulting from the refinement of the root goal. This is a size metric of the whole goal model, and can be used as a surrogate for model complexity. In addition, the auxiliary metric NGSG can be used for any sub-goal to help identifying structural problems in goal decomposition. A goal with too many sub-goals should be scrutinized for a potential lack of cohesion.

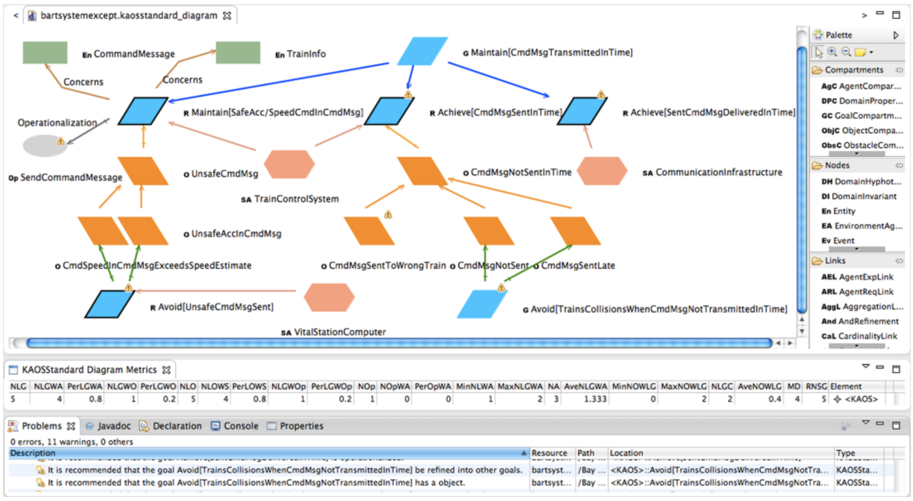
Table 3. Metrics to satisfy the complexity goal – Q6 through Q9

Q6 - Does an agent have too much responsibility in the model?	
Name	**ANLG – N**umber of **L**eaf **G**oals of an **A**gent
Informal definition	Number of direct or indirect leaf goals connected to this agent.
Formal definition	**context Agent** **def: ANLG(): Integer** = self.ALG()->size()
Comments	This requires: **ALG –** Set of all direct or indirect **A**gent **L**eaf **G**oals
Recommendation	Too much responsibility to an agent should be avoided. In general this hints the agent is too generic, so more specialized agents should be considered.
Q7 - Does a leaf goal have too many/few associated objects?	
Name	**GNO – N**umber of **O**bjects of a **G**oal
Informal definition	Number of objects (entities) connected to this goal.
Formal definition	**context Goal** **def: GNO(): Integer** = self.goalConcerns->collect(concernsObject)->size()
Comments:	This is a simple count of objects connected to a goal.
Recommendation	Too many objects associated with a goal should be avoided. In general this is a hint for the need of decomposing a goal into sub-goals.
Q8 - How difficult is understanding a leaf goal, with respect to its parent goals?	
Name	**MD – M**odel **D**epth
Informal definition	Depth of the model, considering goals and obstacles.
Formal definition	**context KAOS** **def: MD(): Integer** = self.NDR(self.root.oclAsType(Nodes), Set{}, Set{0}, 0)->iterate(i: Integer; maxD: Integer = -1 | if maxD = -1 then i else maxD.max(i) endif)
Comments	The model depth includes goals and obstacles. This requires: **NDR – N**odes **D**istance to **R**oot
Recommendation	The deeper the model, the harder it is to understand it, so model depth should be kept to a minimum in terms of how it depends on system size.
Q9 - How complex is a goal, with respect to its refinements?	
Name	**RNSG – R**oot **N**umber of **S**ub-**G**oals
Informal definition	Number of direct or indirect sub-goals of the root.
Formal definition	**context KAOS** **def: RNSG(): Integer** = self.root.NGSG()
Comments	This requires: **NGSG – N**umber of direct or indirect **S**ub-**G**oals of a **G**oal
Recommendation	A goal with too many sub-goals should be scrutinized for potential lack of cohesion.

3.2 Example

We now present the modularKAOS tool[2] through an example (Fig. 1). The tool allows Requirements Engineers to build goal models using a visual language and provides some feedback on the model (such as design warnings and metrics values). These values can be updated at any time, while using the tool. As such, they can be valuable for developers to detect potential problems early in the process (e.g. a high accidental complexity caused by some modeling option) and gauge how close they are from completion. In this sense, the proposed framework should be mostly regarded as a facilitator during the modeling process, rather than as a *post-mortem* analysis tool, although it supports both activities.

Fig. 1 shows a fragment of the Bay Area Rapid Transit System (BARTS) case study [14], whose main objective is to make a train system more efficient by running trains more closely spaced. The goal **Maintain[CmdMsgTransmitted-InTime]** specifies that the command messages must be transmitted in time, to avoid collisions with other trains. This goal represents the case where a message requiring a safe acceleration, based on the speed and position of the following and preceding trains, is exchanged between the train and the control system. This goal is refined into three sub-goals: **Achieve[CmdMsgSentInTime]**, where the command message is sent in time by the **TrainControlSystem** to the train; **Maintain [SafeAcc/SpeedCmdIn-CmdMsg]**, where the **TrainControlSystem** tries to maintain a safe acceleration in the messages that send to the trains; and **Achieve[SentCmdMsgDeliveredInTime]**, where the **CommunicationInfrastructure** guarantees that the messages are sent.



KAOSStandard Diagram Metrics

NLG	NLGWA	PerLGWA	NLGWO	PerLGWO	NLO	NLOWS	PerLOWS	NLGWOp	PerLGWOp	NOp	NOpWA	PerOpWA	MinNLWA	MaxNLGWA	NA	AveNLGWA	MinNOWLG	MaxNOWLG	NLGC	AveNOWLG	MD	RNSG	Element
5	4	0.8	1	0.2	5	4	0.8	1	0.2	1	0	0	1	2	3	1.333	0	2	2	0.4	4	5	<KAOS>

0 errors, 11 warnings, 0 others

Description	Resource	Path	Location	Type
It is recommended that the goal Avoid[TrainsCollisionsWhenCmdMsgNotTransmittedInTime] be refined into other goals.	bartsyst...	/Bay ...	<KAOS>::Avoid[TrainsCollisionsWhenCmdMsgNotTra...	KAOSSta...
It is recommended that the goal Avoid[TrainsCollisionsWhenCmdMsgNotTransmittedInTime] has a object.	bartsyst...	/Bay ...	<KAOS>::Avoid[TrainsCollisionsWhenCmdMsgNotTra...	KAOSSta...

Fig. 1. Application of the modularKAOS tool and the metrics to the BARTS case study

[2] The tool is available at `http://ctp.di.fct.unl.pt/~mgoul/papers/2013/CAiSE2013/modularKAOS/`

The diagram contains some obstacles and respective resolutions. If the obstacles have no direct or indirect solutions, a warning is thrown (e.g., in the **Avoid** resolution requirement). Whenever leaf goals are neither linked to an object nor to an operation a warning is thrown (e.g., in the **Avoid** leaf goal). The operation presented in the model throws a warning as it should be performed by an agent and no agent is assigned to it.

The warnings are listed in the problems tab below the metrics tab presenting the metrics values of this model. We have 5 leaf goals (NLG = 5 - **Maintain[SafeAcc/SpeedCmdInCmd-Msg]**, **Avoid[TrainsCollisionsWhenCmdMsg-NotTransmittedInTime]**, **Achieve[CmdMsgSentInTime]**, **Achieve[Unsafe Cmd-MsgSent]**, **Achieve[SentCmdMsgDeliveredInTime]**), of which 4 have an agent (NLGWA = 4 – first four goals listed above), and so on. Note that some of the auxiliary metrics, presented in this screenshot, are not discussed in this paper (e.g., MaxNLGWA – the maximum number of leaf goals with an agent in the model), but are defined in the previously mentioned complete list of metrics.

4 Evaluation

4.1 Case Studies

We modelled well-known KAOS real world case studies, namely the Bay Area Rapid Transit (BARTS) [14], the London Ambulance Service (LAS) [14], the Elevator System (ES) [15], the Meeting Scheduler (MS) [2], the Library Management System (LMS) [2], the Mine Safety Control System (MSCS) [2], and the Car Park Management System (CPMS) [16] with modularKAOS, and then collected the corresponding metrics. These case studies are described as part of a text book [2], a PhD dissertation [14] and tutorials [15, 16], so they provide enough detail, in contrast with what could happen with examples taken from papers published in conferences and, to a lesser extent, in journals.

4.2 Results and Discussion for the Completeness Metrics

For each question related to the completeness goal, we present a column chart. Each column is associated to a different system, on the left side of the picture, and a box-plot chart, with the dispersion, skewness and outliers, on the right side of the picture.

Q1) PLGWA: This question concerns how far a model is from assigning all its goal responsibilities to agents. On the left side of Fig. 2.a), we see the percentage of leaf goals with an agent for each goal model of each case study. The CPMS has the most complete model, concerning goal responsibility assignment, while BARTS has a much lower percentage than the other case studies. BARTS is an outlier in the boxplot chart, indicating a significantly lower focus on agent's responsibilities assignment in this system. Overall, around 70% of the leaf goals in the goal models follow the completeness rule that specifies that a leaf goal should be assigned to an agent.

Q2) PLGWO: The second question concerns the level of detail of a goal model with respect to the percentage of leaf goals with an object. Fig. 2.b) shows that most case studies barely specify objects in their goal models. BARTS and LASS provide more leaf goals with objects in the model, although with a percentage lower than 50%.

CPMS, LMS and MSS do not represent objects at all. In the boxplot the median is below 10%, but no outliers are identified. These case studies seem to indicate that object identification was not regarded as a priority at this requirements stage.

Q3) PLOWS: The third question concerns the goal model level of detail with respect to obstacles that have resolutions. Fig. 2.c) shows that the CPMS and ES (both taken from KAOS tutorials) are the ones that provide more obstacles with resolutions. The remaining case studies have a lower PLOWS, suggesting that the specification of a resolution for obstacles is not a major concern at this requirements stage. LMS is a missing value here, because there are no leaf obstacles in this model (and, therefore, no resolution for them). With an even number (six) of observations, there is no single middle value and the median is the mean of the two middle values [17].

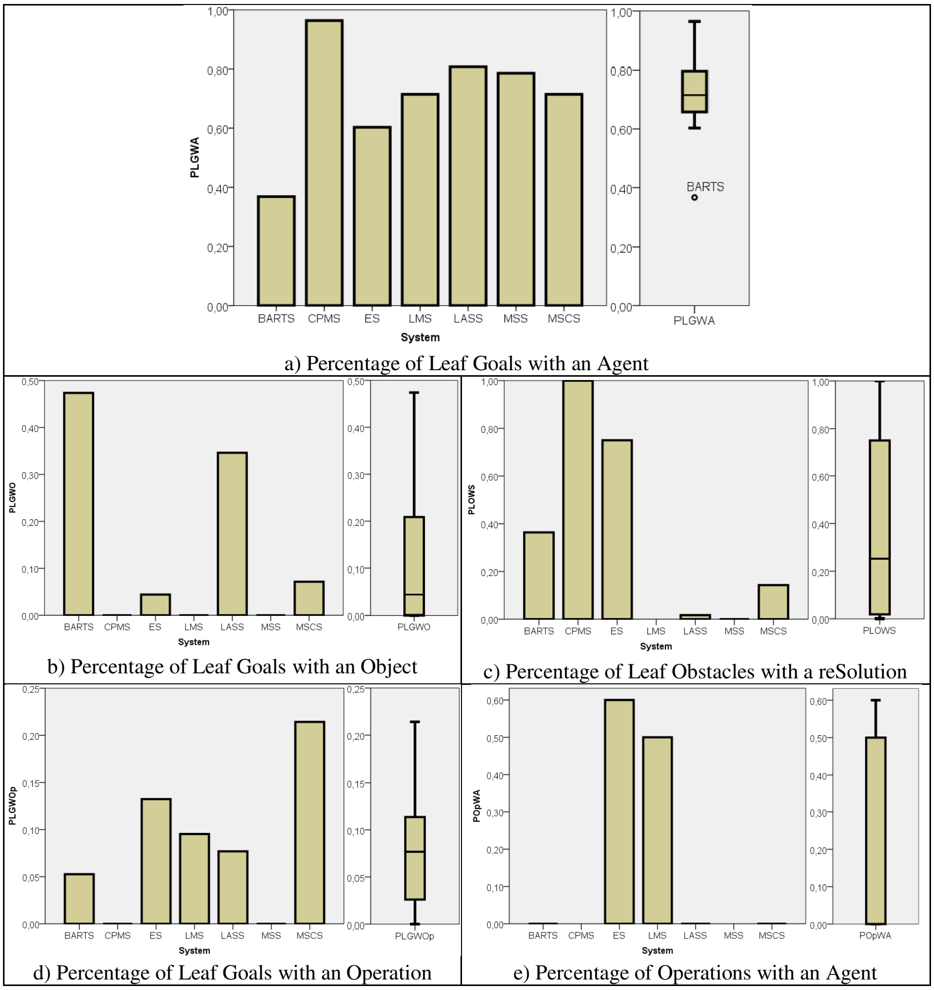


Fig. 2. Metrics values for our case studies

Q4) PLGWOp: The fourth question concerns the level of detail of a goal model with respect to operations associated to leaf goals. Fig. 2.d) shows that operations were rarely specified in the goal models of the case studies with a rate of leaf goals with operation lower than 25%. MSCS is the case study where operations appear more frequently. The median shown in the boxplot is below 10%, but no outliers are identified. Operations identification and association to leaf goals do not seem to be a major concern at this stage.

Q5) POpWA: The fifth question concerns the level of detail of a goal model with respect to operations associated to agents. Note that, as seen in Fig. 2.d), CPMS and MSS have no operations specified. Therefore, they are missing values, as shown in Fig. 2.e) (it does not make sense to compute the POpWA metric on them, as specified in the pre-condition shown in Table 2). Three case studies (BARTS, LASS and MSCS) out of the remaining five have no operations assigned to an agent. Only ES and LMS use this feature, and do so for around half of the operations. These results suggest a lower emphasis on the complete specification of operations in the goal models, regarding their assignment to agents.

4.3 Results for the Metrics Related to the Complexity Goal

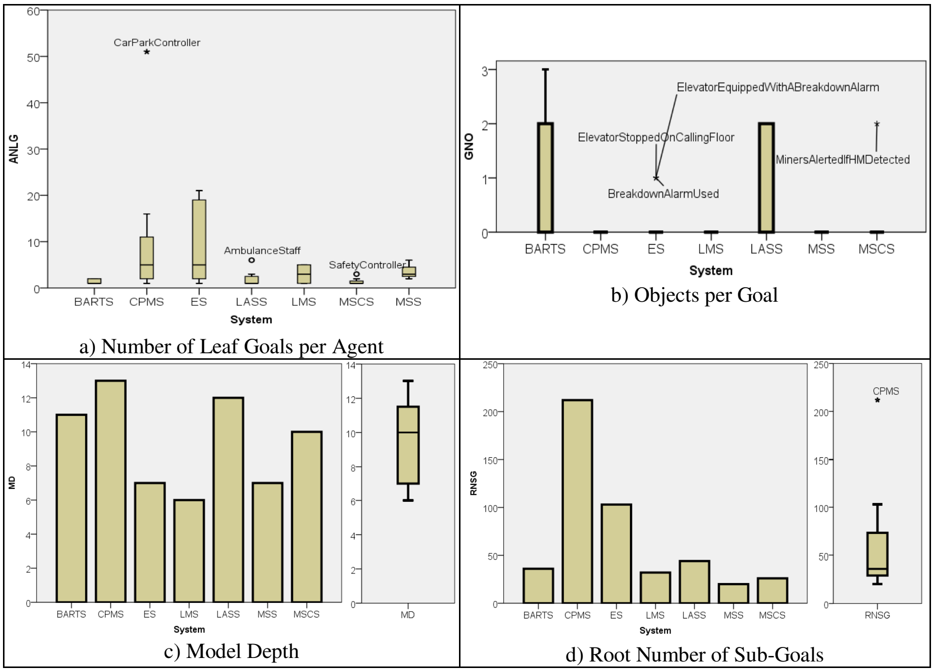
For the first two questions related to the complexity goal we show only a boxplot chart (since percentages are not involved, only discrete values). For the last two questions we show both a column and a boxplot chart.

Q6) ANLG: Question six concerns the degree of responsibility of an agent. As seen in Fig. 3.a), most models have between 1 and 5 leaf goals per agent. The two case studies taken from tutorials (CPMS and ES) have a significantly higher number of leaf goals per agent. The most extreme case has a very high ANLG suggesting too many responsibilities for the CarParkController agent. This would be a candidate for further agent decomposition.

Q7) GNO: Question 7 concerns the number of objects associated with a goal. Fig. 3.b) shows how the majority of the case studies do not consider objects in the goal models. BARTS and LASS are the exceptions where objects are more frequently used. The extreme values in the ES and MSCS case studies, with one object per goal, confirm how rarely these model elements are used in those case studies.

Q8) MD: Question 8 concerns the level of understandability of a goal model, based on its depth (using a complexity metric similar to the one proposed in [18] for object-oriented design). Fig. 3.c) shows that, in our sample, the most complex system is CPMS, with 13 levels of refinement. However, the median shown in the boxplot reveals that the number of refinement levels is around 10 and no outliers were identified, suggesting a fairly consistent number of decomposition levels is used. Extreme values of this metric could indicate variations in the accidental complexity of models. A much lower level of MD could suggest a simplistic structuring of the model, while extremely high values could hint for possible “over-refinement” of the model.

Q9) RNSG: Question 9 addresses the essential complexity evaluation by considering the total number of model elements (in particular, the number of subgoals). Fig. 3.d) shows that CPMS has the highest number of goals and the boxplot shows it as an outlier, with more than 200 subgoals, while the median considering all the case studies is below 50. This suggests that the CPMS case study itself is defined with more details than the other ones.



a) Number of Leaf Goals per Agent
b) Objects per Goal
c) Model Depth
d) Root Number of Sub-Goals

Fig. 3. Metrics values for our case studies

4.4 Discussion on the Metrics Definitions

Concerning our completeness metrics, all of them are defined in such a way that a percentage of the artifacts present in the model, against their potential maximum number for the same model, is computed. The recommendations presented in Table 2 closely follow the guidelines generally accepted by the KAOS modeling community (see, for example, [2]). They can be used as metrics-based heuristics to hint developers to potential incompleteness problems in their model which may be detectable by analyzing the structure of the KAOS goal model. As such, we consider them adequate for the task of assessing model completeness in their respective perspective.

There are, of course, other important completeness perspectives which are not detectable through this approach. The completeness of a goal model also depends on aspects such as the quality of the requirements definition phase. If a goal is not identified during the interviews with stakeholders, it will not be represented in the corresponding KAOS goal model. So, the goal coverage rate depends mostly on those interviews and on the thoroughness of validation meetings where the stakeholders review the proposed goal models, to address potential conflicts and, possibly identify goals that were missed in a previous phase of the requirements elicitation process. Another technique that helps mitigating difficulties in the production of complete requirement models is to use pattern-based refinement techniques that support completeness and consistency of the goal models.

Concerning complexity, a more detailed discussion is required. First of all, it is important to note that models have two sorts of complexity: the essential complexity, which is intrinsic to the system being modeled, and the accidental complexity, which results from the way the system is being modeled (including the chosen modeling approach). In general, this accidental complexity should be minimized, if possible.

To the best of our knowledge, there are no guidelines concerning what is considered an "acceptable" level of complexity in KAOS goal models. However, there is a large body of work concerning metrics-based model complexity evaluation for other classes of software models. In particular, Object-Oriented model complexity evaluation has been often addressed using software metrics (e.g. [18]). A common way of using complexity metrics to help developers, is to collect such metrics from models generally considered as good design examples (or, conversely, from bad ones). The rationale is that, if a sufficiently large body of examples can be collected, typical values can then be identified, in order to support design heuristics. Complexity metrics can be used to help detecting bad design smells (e.g. the presence of a god class), or bad code smells (e.g. the presence of a long method), when "unusual values" are detected.

In this paper, we adapt this idea to the context of goal models. Although a larger number of case studies would be necessary in order for these typical values of a good (or bad) design to start emerging with statistical significance, we can already have a first look at some modeling tendencies from these case studies. We can also identify some modeling issues that might require further attention. For example, the extremely high number of leaf goals per agent assigned to the `CarParkController` agent, in figure 3.a, would be a likely candidate for a closer analysis, considering how different this number is, when compared to the number of leaf goals assigned to other agents in the rest of this system (and, in fact, in all the other case studies in this paper). For each of the proposed complexity metrics, we briefly present a hands-on recommendation, in table 3, alerting to the potential problem in the goal model that the metric helps identifying. Note that although the "unusual value" of the complexity metric is a hint for a potential problem in the goal model, this has then to be checked. It may be the case that a high complexity value results from the essential complexity of the problem, rather than from a less than optimal model.

The body of work in software complexity metrics includes approaches to their validation. Weyuker proposed a set of 9 desirable properties for program complexity metrics [19]. These were frequently adopted for assessing metrics sets not only in programs, but also in design documents such as class diagrams. We adapt those definitions to the context of complexity metrics for GORE models.

Consider P, Q, and R as models. Let $|P|$, $|Q|$, and $|R|$ be their complexity, respectively, as measured by a complexity metric. Let $|P; Q|$ be the resulting complexity of P composed with Q. Table 4 presents the adapted Weyuker properties, in natural language and formally, and identifies which metrics satisfy those properties.

Most of these properties for complexity metrics are preserved by our complexity metrics. Property 9 is a noticeable exception. We assume that the composition of two models does not introduce model elements which are not present in at least one of the original models. With these metrics, a composed model cannot exhibit higher metrics values than the sum of its parts. Without this assumption, property 9 would hold for these metrics. RNSG does not preserve property 7. A reorganization of the model that does not introduce new model elements does not change the number of sub-goals in the model. It is common to find high impact complexity metrics (e.g. [18]) that do not fulfill all of these “desirable” properties (see, for instance, [20]).

Table 4. Adapted Weyuker Properties

#	Adapted Weyuker Property	ANLG	GNO	MD	RNSG
1	At least some different models should exhibit different values for the same complexity metric.	Yes	Yes	Yes	Yes
	$\exists P, \exists Q : P \neq Q \land \lvert P \rvert \neq \lvert Q \rvert$				
2	There is a finite number n of models for which the complexity is c (a non-negative number). Let S be the set of models with c complexity, and n the cardinal of the set S.	Yes	Yes	Yes	Yes
	$\forall c \in \mathbb{R}_0^+, \forall P : \lvert P \rvert = c \Rightarrow P \in S, \exists n \in N_0 : \#S = n$				
3	Different models P and Q may have the same complexity.	Yes	Yes	Yes	Yes
	$\exists P, \exists Q : P \quad Q \land \lvert P \rvert = \lvert Q \rvert$				
4	Different models which are functionally equivalent may have different complexities.	Yes	Yes	Yes	Yes
	$\exists P, \exists Q : P \equiv Q \land \lvert P \rvert \neq \lvert Q \rvert$				
5	Monotonicity is a fundamental property of all complexity measures. A model in isolation is at most as complex as its composition with another model.	Yes	Yes	Yes	Yes
	$\forall P, \forall Q : \lvert P \rvert \leq \lvert P; Q \rvert \land \lvert Q \rvert \leq \lvert P; Q \rvert$				
6	The resulting complexities of composing the same model (R) with two different models of the same complexity (P and Q) are not necessarily equal. Conversely, the complexities of composing two different models (P and Q) of the same complexity with a third program (R) are also not necessarily equal.	Yes	Yes	Yes	Yes
	$\exists P, \exists Q, \exists R : P \neq Q \land \lvert P \rvert = \lvert Q \rvert \land \lvert P; R \rvert \neq \lvert Q; R \rvert$ $\exists P, \exists Q, \exists R : P \neq Q \land \lvert P \rvert = \lvert Q \rvert \land \lvert R; P \rvert \neq \lvert R; Q \rvert$				
7	Weyuker’s property 7 states that program’s complexity should be responsive to the order of its statements, and hence to their potential interaction. In a KAOS goal model, the adapted rule would be that the model complexity should be responsive to the organization of its model elements in the goal model graph. Let P be a model and Q another model such that Q is formed by permuting the order of the elements in P. Assume we name this permutation operation Perm().	Yes	Yes	Yes	No
	$\exists P, \exists Q : Q = Perm(P) \land \lvert P \rvert \neq \lvert Q \rvert$				
8	If a model is a renaming of another model, then their complexity should be the same. Assume that the operation Rename() transforms program P in its renamed version Q.	Yes	Yes	Yes	Yes
	$\forall P, \forall Q : Q = Rename(P) \Rightarrow \lvert P \rvert = \lvert Q \rvert$				
9	The complexity of the composition of two models P and Q may be greater than the sum of the complexities of models P and Q. The extra complexity may result from the interaction between programs P and Q.	No	No	No	No
	$\exists P, \exists Q : \lvert P \rvert + \lvert Q \rvert < \lvert P; Q \rvert$				

4.5 Summary

Concerning completeness, in our sample of KAOS models: (i) most models handle responsibility assignment of leaf goals to agents; (ii) objects are not frequently used in the case studies; (iii) when obstacles are specified, we find a big variation (from 0% to 100%) of the percentage of obstacles with a resolution, suggesting the concern for specifying obstacle resolutions is not consistently spread among the requirements engineers (some may prefer to postpone this to later stages); (iv) operations are even more rarely used than objects – again, this seems to point to a preference for postponing their definition to later stages; (v) only two of the case studies model the assignment of operations to agents, showing this is a fairly unexplored modeling feature.

Concerning the complexity goal, we found that: (i) in most cases, the number of leaf goals assigned to an agent is relatively small, indicating, with few exceptions, a general concern of not attributing too many responsibilities to a single agent; (ii) assigning objects to goals is a mostly unexplored feature of models (iii) model depth varies much less than the number of model elements, suggesting a fairly consistent state of practice with respect to what is considered an adequate model decomposition level; (iv) we found big variations in the case studies, concerning the number of subgoals defined in each model; although the average number is around 40 subgoals, in one of the examples it is over 200 goals. A closer inspection of the CPMS model showed that the main source of variation was the significantly higher level of detail with which this model was built.

4.6 Validity Threats

The case studies used in this paper are generally considered as good examples of real-world KAOS models and can be, in that sense, used as a reference for best practices in goal modeling. However, other industry-based specifications may show different profiles of utilization of the modeling mechanisms. Nevertheless, for the purposes of validating the proposed metrics set, our sample of case studies covers all the situations we are addressing with this metrics set.

5 Related Work

Ramos et al. claim that early identification of syntactical problems (e.g., large and unclear descriptions, duplicated information) and the removal of their causes can improve the quality of use case models [19]. They describe the AIRDoc approach, which aims to facilitate the identification of potential problems in requirements documents using refactoring and patterns. To evaluate use case models, the AIRDoc process uses the GQM approach to elaborate goals and define questions to be addressed by metrics. Their target quality attributes are reusability and maintainability, different from ours. Their metrics were neither formally defined nor implemented in a tool.

Vasconcelos et al. [20] claim that GORE and MDD can be integrated to fulfill the requirements of a software process maturity model in order to support the application of GORE methodologies in industry scenarios. The proposed approach, called GO-MDD, describes a six-stage process that integrates the i* framework into a concrete

MDD process (OO-Method), applying the CMMi perspective. The fourth stage of this process concerns the verification, analysis and evaluation of the models defined in the previous stages; and uses a set of measurements, specified with OCL rules, that evaluate the completeness of the MDD model generation with respect to the requirements specified in the i* model. The set of metrics used in this stage is presented in [21], using GQM. Compared to ours, their approach focuses on a different set of metrics as their goal was to support the evaluation of i* models to generate MDD models.

Franch and Grau in [22] propose a framework for defining metrics in i* models, to analyze the quality of individual models, and to compare alternative models over certain properties. This framework uses a catalogue of patterns for defining metrics, and OCL to formulate these metrics. In a follow up work, Franch proposes a generic method to better guide the analyst throughout the metrics definition process, over i* models [23]. The method is applied to evaluate business process performance.

6 Conclusions and Future Work

In this paper, we proposed a metrics suite for evaluating the completeness and complexity of KAOS goal models, formally specified (using OCL) and incorporated in a DSL based modeling tool. In the context of large-scale systems, neglecting completeness at early stages may ultimately lead to unexpected extra costs in the later development stages. Moreover, completeness analysis is useful to help requirements engineers to evaluate how close they are to completing their models. Complexity analysis is particularly useful for identifying issues with the quality of the produced models. In particular, it can be used to help identifying opportunities for requirements refactoring. Leveraging tool-supported completeness and complexity evaluation contributes to a deeper understanding of requirements models and can be used to enhance the overall quality of those models. We validated these metrics by applying them to several real-world case studies that are generally considered as good examples of GORE. As such, the obtained metrics values mirror a pattern of usage in goal modeling.

In a near future, we intend to extend the metrics set to cover other model quality attributes and replicate this evaluation with other KAOS models. This would be a stepping stone towards integrating metrics-based modeling heuristics in GORE tools. We also plan to address completeness in terms of requirements coverage, by tracing the model elements to requirements sources and identifying the requirements in those sources that are yet to be covered by the goal models.

Acknowledgments. The authors would like to thank the AMPLE project and CITI – PEst-OE/EEI/UI0527/2011, and to FFCT/UNL and the ASSD project, supported by ESA, for the financial support for this work, and Robert Darimont for providing us the documentation on the CPMS case study.

References

1. Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: 5th IEEE International Symposium on Requirements Engineering, pp. 249–262. IEEE (2001)
2. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley (2009)

3. Yu, E.: Modelling strategic relationships for process reengineering. Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada (1995)
4. Anton, A.I.: Goal-Based Requirements Analysis. In: 2nd International Conference on Requirements Engineering (ICRE 1996). IEEE (1996)
5. ITU-T: Recommendation Z.151 (09/08): User Requirements Notation (URN)–Language definition. International Telecommunications Union, Geneva, Switzerland (2008)
6. Brooks, F.P.: No silver bullet: essence and accidents of software engineering. IEEE Computer 20, 10–19 (1987)
7. Espada, P., Goulão, M., Araújo, J.: Measuring Complexity and Completeness of KAOS Goal Models. In: International Workshop on Empirical Requirements Engineering, EmpiRE 2011. IEEE (2011)
8. OMG: OMG Object Constraint Language (OCL). Object Management Group (2012)
9. Dias, A., Amaral, V., Araújo, J.: Towards a Domain Specific Language for a Goal-Oriented Approach based on KAOS. In: Third International Conference on Research Challenges in Information Systems (RCIS 2009). IEEE (2009)
10. Lamsweerde, A., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering 26, 978–1005 (2000)
11. Matulevicius, R., Heymans, P.: Analysis of KAOS Meta-model. University of Namur, Computer Science Department, Belgium (2005)
12. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. Addison-Wesley Professional (2003)
13. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineeringm, pp. 528–532. Wiley (1994)
14. Letier, E.: Reasoning about Agents in Goal-Oriented Requirements Engineering. PhD Thesis, Catholic University of Louvain, Belgium (2001)
15. Respect-IT: A KAOS Tutorial, version 1.0 (2007)
16. Respect-IT:e-Learn GORE & Objectiver, http://www.objectiver.com/index.php?id=108
17. Weisstein, E.W.: Statistical Median - MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/StatisticalMedian.html
18. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering 20, 476–493 (1994)
19. Ramos, R., Castro, J., Araújo, J., Moreira, A., Alencar, F., Santos, E., Penteado, R.: AIRDoc–An Approach to Improve Requirements Documents. In: 22nd Brazilian Symposium on Software Engineering, SBES 2008 (2008)
20. de Vasconcelos, A.M.L., Giachetti, G., Marín, B., Pastor, O.: Towards a CMMI-Compliant Goal-Oriented Software Process through Model-Driven Development. In: Johannesson, P., Krogstie, J., Opdahl, A.L. (eds.) PoEM 2011. LNBIP, vol. 92, pp. 253–267. Springer, Heidelberg (2011)
21. Giachetti, G., Alencar, F., Franch, X., Pastor, O.: Applying i* Metrics for the Integration of Goal-Oriented Modeling into MDD Processes. Universitat Politècnica de Catalunya, Barcelona, Spain (2010)
22. Franch, X., Grau, G.: Towards a Catalogue of Patterns for Defining Metrics over *i** Models. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 197–212. Springer, Heidelberg (2008)
23. Franch, X.: A Method for the Definition of Metrics over *i** Models. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 201–215. Springer, Heidelberg (2009)