Identifying Modularity Improvement Opportunities in Goal-Oriented Requirements Models

Catarina Gralha, Miguel Goulão, and João Araújo

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

Abstract. Goal-oriented Requirements Engineering approaches have become popular in the Requirements Engineering community as they provide expressive model elements for requirements elicitation and analysis. However, as a common challenge, they are still struggling when it comes to managing the accidental complexity of their models. In this paper, we provide a set of metrics, which are formally specified and have tool support, to measure and analyze the complexity of goal models, in particular *i** models. The aim is to identify refactoring opportunities to improve the modularity of those models, and consequently reduce their complexity. We evaluate these metrics by applying them to a set of well-known case studies from industry and academia. Our results allow the identification of refactoring opportunities in the evaluated models.

Keywords: Goal-Oriented Requirements Models, *i**, software metrics, model assessment.

1 Introduction

Goal-oriented Requirements Engineering (GORE) has a great impact and importance in the Requirements Engineering community, helping in identifying, organizing, and structuring requirements, as well as in exploring and evaluating alternative solutions to a problem [1]. There are several GORE approaches, such as *i** [2], KAOS [3], and GRL [4]. When modelling real-world systems with a GORE approach, the models can quickly become very complex. A common challenge for the GORE approaches is to manage the complexity of their models. While real-world problems have an unavoidable essential complexity, we need to minimize, as much as possible, the accidental complexity introduced by the way we model those problems [5].

A possible way of minimizing the accidental complexity of a model is to improve its modularity. In particular, this can be achieved by identifying model refactoring opportunities. In this paper, we focus on the *i** framework, and how we can manage the accidental complexity of *i** models. In order to identify refactoring opportunities for these models, we define a metrics suite for assessing their complexity and the complexity of the elements defined in them. By collecting such metrics on several different models, we are able to establish a typical usage profile of the modelling mechanisms.

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 91–104, 2014.

In practice, this profile is built using descriptive statistics analysis on the metrics collected from different model elements. For example, the number of goals and tasks for a system agent may indicate whether this agent holds too many responsibilities in the system. This can hint the modeler for a refactoring opportunity where this agent should in fact be decomposed into several sub-agents.

The objective of this paper is to provide a metrics suite, along with the corresponding tool support, targeted to the measurement and analysis of the complexity of *i** models, with the goal of identifying refactoring opportunities to improve the modularity of those models. The identification of such opportunities can be useful during the development of the system, where a better modularization can lead to a sounder distribution of responsibilities among the system components. If performed in a timely fashion, this is likely to contribute to relevant costs savings through the reduction of the model's accidental complexity. Refactoring opportunities identification is also an asset in the context of preventive maintenance, as a facilitator for future requirements changes.

Our metrics suite is integrated in an eclipse-based *i** editor, so that metrics can be computed during the requirements modelling process, whenever the requirements engineer requests them. The metrics are defined using the Object Constraint Language (OCL) [6] upon the *i** metamodel. This makes our metrics set easily extensible, as improving the metrics set can be done by adding new OCL metrics definitions to the ones presented in this paper.

In [7], we proposed and validated a metrics suite for evaluating the completeness and complexity of KAOS goal models, formally specified (using OCL) and incorporated in a KAOS modelling tool. The metrics suite was evaluated with several real-world case studies. The work described in this paper shares a common approach to metrics definition and tool implementation. However, the goals and structure of the KAOS approach are significantly different from those of the *i** framework. In particular, *i** has a modularity mechanism – the actor's boundaries – which is not present in KAOS, that paves the way for a significantly different approach to modularity, by encapsulating model elements within the actors boundaries. This is reflected in the choice of relevant complexity metrics. Actor's boundaries are a key mechanism in the metrics suite proposed in this paper. Our goal is to use these metrics to leverage the modularity of *i** models.

This paper is organized as follows. Section 2 describes background information on the *i** framework. Section 3 describes the metrics set, defined using the Goal-Question-Metrics 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. While the paper is self-contained, additional information such as the complete *i** metamodel, the detailed specification of auxiliary metrics, and the fully detailed statistical analysis of the case studies presented in this paper can be found in this paper's companion site[1].

[1] http://ctp.di.fct.unl.pt/~mgoul/CAiSE2014Companion/

2 The *i** Approach

*i** [2] was developed for modelling and reasoning about organizational environments and their information systems. It focuses on the concept of *intentional actor*. Actors in their organizational environment are viewed as having intentional properties such as *goals*, *beliefs*, *abilities* and *commitments*. *i** has two main modelling components: the *Strategic Dependency* (SD) model and the *Strategic Rationale* (SR) model. The SD model describes the dependency relationships among the actors in an organizational context. In this model, an actor (called *depender*) depends on another actor (called *dependee*) to achieve *goals* and *softgoals*, to perform *tasks* and to obtain *resources*. The SR model provides a more detailed level of modelling than the SD model, since it focuses on the modelling of intentional elements and relationships internal to actors. Intentional elements (*goals*, *softgoals*, *tasks* and *resources*) are related by *means-end* or *decomposition* links. *Means-end links* are used to link *goals* (ends) to *tasks* (means) in order to specify alternative ways to achieve goals. *Decomposition links* are used to decompose tasks. A task can be decomposed into four types of elements: a *subgoal*, a *subtask*, a *resource*, and/or a *softgoal*. Apart from these two links, there are the *contribution links*, which can be *positive* or *negative*.

In this work we are particularly interested in assessing the complexity of *i** models. To support this, we needed a flexible platform upon which we could define our metrics set. To the best of our knowledge, none of the existing *i** tools provides adequate support for a flexible definition of such metrics (detailed comparison of the existing *i** tool support can be found in [8, 9]). One of the important requirements of the tool was that it should be extensible, so that new metrics (which can potentially target different quality attributes) can be easily added. To fill this gap, we implemented an Eclipse-based *i** editor using Epsilon [10], EMF/GMF [11, 12] and Ecore Tools [13].

Figure 1 presents a fragment of the *i** metamodel implemented in our tool, showing only the concepts which will be used in the metrics definitions proposed in this paper. This metamodel is the basis for the tool support for the specification of *i** models, and their evaluation with model metrics. The root of the metamodel is the

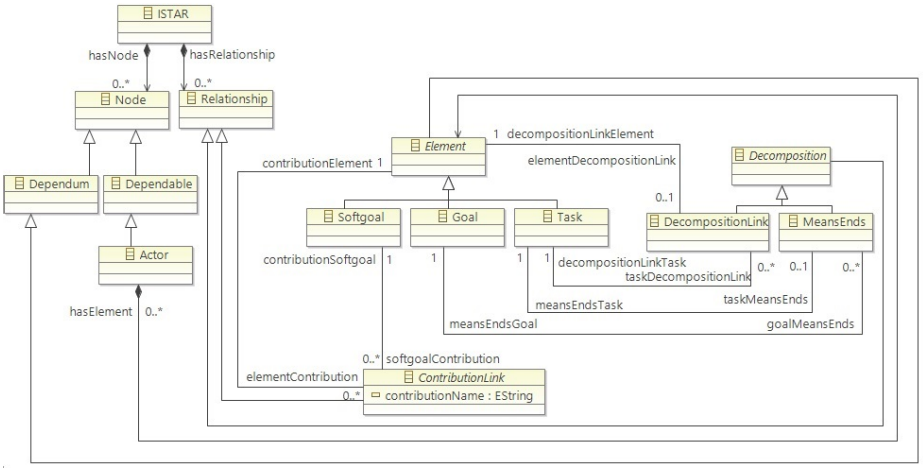


Fig. 1. Partial *i** metamodel

metaclass *ISTAR*, which contains all the nodes and relationships of an *i** model. This top-level metaclass serves as a basis for model analysis. The remaining metaclasses can be easily mapped into some of the concepts described earlier in this section.

3 A Metrics Set for *i** Complexity Evaluation

The purpose of this study is to evaluate the complexity of *i** models. We propose a metrics-based analysis framework for *i** models, using the Goal-Question-Metric (GQM) approach [14]. Table 1 summarizes the GQM-based proposal for a set of metrics that will allow satisfying the goal of *complexity evaluation*. The first column presents *questions* that will allow evaluating whether the overall goal is being achieved. The second column shows a set of *metrics* that provide quantitative information to answer the corresponding question. ***Q1*** concerns complexity, as perceived when regarding the model as a whole. In particular, we are interested in the number of actors, elements, and their ratio, within a model. The remaining questions are targeted to assessing the complexity of model elements, namely the amount of responsibilities supported by an actor (***Q2***), and the number of decompositions of actor's goals (***Q3***), softgoals (***Q4***) and tasks (***Q5***). For each of these elements-centered questions, we define a basic metric (e.g. *NEA*, for ***Q2***) and three additional distribution metrics presenting the *minimum*, *maximum* and *average* values for the basic metric.

Table 1. Goal-Question Metric for *i** models evaluation

Goal: Complexity evaluation	
Question	**Metric**
***Q1* –** How complex is the model, concerning the number of actors and elements?	**NAct** – **N**umber of **Act**ors **NElem** – **N**umber of **Elem**ents
***Q2* –** Does an actor have too much responsibility in the model?	**NEA** – **N**umber of **E**lements of an **A**ctor **MinNEA** – **Min**imum **N**umber of **E**lements of an **A**ctor **MaxNEA** – **Max**imum **N**umber of **E**lements of an **A**ctor **AvgNEA** – **Av**erage **N**umber of **E**lements of an **A**ctor
***Q3* –** How complex is an actor's goal, with respect to its decompositions?	**NDG** – **N**umber of **D**ecompositions of an actor's **G**oal **MinNDG** – **Min**imum **N**umber of **D**ecompositions of an actor's **G**oal **MaxNDG** – **Max**imum **N**umber of **D**ecompositions of an actor's **G**oal **AvgNDG** – **Av**erage **N**umber of **D**ecompositions of an actor's **G**oal
***Q4* –** How complex is an actor's softgoal, with respect to its decomposition?	**NDS** – **N**umber of **D**ecompositions of an actor's **S**oftgoal **MinNDS** – **Min**imum **N**umber of **D**ecompositions of an actor's **S**oftgoal **MaxNDS** – **Max**imum **N**umber of **D**ecompositions of an actor's **S**oftgoal **AvgNDS** – **Av**erage **N**umber of **D**ecompositions of an actor's **S**oftgoal
***Q5* –** How complex is an actor's task, with respect to its decompositions?	**NDT** – **N**umber of **D**ecompositions of an actor's **T**ask **MinNDT** – **Min**imum **N**umber of **D**ecompositions of an actor's **T**ask **MaxNDT** – **Max**imum **N**umber of **D**ecompositions of an actor's **T**ask **AvgNDT** – **Av**erage **N**umber of **D**ecompositions of an actor's **T**ask

3.1 Metrics Definition

In Table 2 we present the metrics outlined in the previous section. For each question we present an informal definition of the metrics specified to answer it, and a formal

definition using OCL upon the metamodel fragment presented in figure 1. When required, we include pre-conditions in the formal definition. For example, when defining metrics to compute the average decomposition of *goals*, *softgoals*, or *tasks*, a typical pre-condition is to ensure that there are *goals*, *sofgoals*, or *tasks*, to be decomposed. Elements without decompositions may have been modeled in order to be final elements. It would not make sense analyzing the extent to which they are decomposed. For the sake of brevity, we omit trivial auxiliary metrics definitions with basic counts in the paper. The full metrics suite definition in OCL, including all auxiliary metrics, can be found in the paper's companion site.

Regarding question ***Q1***, the values of *NAct* (number of actors) and *NElem* (number of elements) are measures for the SD/SR model size. Size can be used as a surrogate for overall model complexity, and used to compare the complexity among different models. For example, different candidate models for the same system can be compared, using these metrics, with respect to their overall complexity.

Concerning question ***Q2***, a high value for *NEA* (number of elements of an actor) can be an indicator that a particular actor has too much responsibility. The *minimum*, *maximum* and *average* values help the requirements engineer recognizing cases where the responsibility is higher than expected. Complexity can also be used for supporting project estimation efforts.

Questions ***Q3***, ***Q4*** and ***Q5***, provide different perspectives on the complexity associated with a particular actor. The value of *NDG* (number of decompositions of an actor's goal), presented in ***Q3***, measures the complexity of the *goal* decompositions associated with an *actor*. The value of *NDS* (number of decompositions of an actor's softgoal), presented in ***Q4***, measures the complexity of the *softgoal* decompositions associated with an *actor*. Finally, the value of *NDT* (number of decompositions of an actor's task), presented in ***Q5***, measures the complexity of the *task* decompositions associated with an *actor*. The *minimum*, *maximum* and *average* values for *NDG*, *NDS* and *NDT* help the requirements engineer identifying out of the ordinary *goal*, *softgoal*, or *task* decomposition complexities, respectively. Note that the minimum value is computed only for goals, softgoals, or tasks, which are decomposed. As such, it excludes leaf elements in its computation.

Table 2. Metrics to satisfy the complexity goal

Q1 – Is there a suitable number of actors and elements in the model?	
Name	**NAct** – **N**umber of **Act**ors
Informal definition	Number of actors in the SD/SR model
Formal definition	context ISTAR def:NAct():Integer = self.hasNode -> select (n : Node | n.oclIsKindOf(Actor)) -> size()
Name	**NElem** – **N**umber of **Elem**ents
Informal definition	Number of elements in the SD/SR model
Formal definition	context ISTAR def:NElem():Integer = self.NEOAB() + self.NEIAB()
Requires	**NEOAB** – **N**umber of **E**lements **O**utside **A**ctors' **B**oundaries (see companion site) **NEIAB** – **N**umber of **E**lements **I**nside **A**ctors' **B**oundaries (see companion site)

Table 2. (*Continued.*)

Q2 – Does an actor have too much responsibility in the model?	
Name	**NEA** - **N**umber of **E**lements of an **A**ctor
Informal definition	Number of elements inside an actor's boundary in the SR model
Formal definition	context Actor def:NEA():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Element)) -> size()
Name	**MinNEA** – **Min**imum **N**umber of **E**lements of an **A**ctor
Informal definition	Minimum number of elements inside an actor's boundary in the SR model
Formal definition	context ISTAR def:MinNEA():Integer = self.hasNode -> select(n : Node | n.oclIsKindOf(Actor)) -> iterate(n : Node; min : Integer = -1 | let aux : Integer n.oclAsType(Actor).NEA() in if min = -1 then aux else min.min(aux) endif)
Name	**MaxNEA** – **Max**imum **N**umber of **E**lements of an **A**ctor
Informal definition	Maximum number of elements inside an actor's boundary in the SR model
Formal definition	context ISTAR def:MaxNEA():Integer = self.hasNode -> select(n : Node | n.oclIsKindOf(Actor)) -> iterate(n : Node; max : Integer = -1 | let aux : Integer = n.oclAsType(Actor).NEA() in if max = -1 then aux else max.max(aux) endif)
Name	**AvgNEA** – **Av**era**g**e **N**umber of **E**lements of an **A**ctor
Informal definition	Average number of elements inside an actor's boundary in the SR model
Formal definition	context ISTAR pre:self.NAct() > 0 context ISTAR def:AvgNEA():Real = self.NEA() / self.NAct()
Requires	**NEA** – **N**umber of **E**lements of an **A**ctor **NAct** – **N**umber of **Act**ors

Q3 – How complex is a goal, with respect to its decompositions?	
Name	**NDG** – **N**umber of **D**ecompositions of an actor's **G**oal
Informal definition	Number of decompositions associated with a goal in the SR model
Formal definition	context Goal def:NDG():Integer = self.goalMeansEnds -> select(me : MeansEnds | me.oclIsKindOf(MeansEnds)) -> size()
Name	**MinNDG** – **Min**imum **N**umber of **D**ecompositions of an actor's **G**oal
Informal definition	Minimum number of decompositions associated with a goal that is inside an actor's boundary in the SR model
Formal definition	context Actor def:MinNDG():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Goal) and e.oclAsType(Goal).NDG() > 0) -> iterate(e : Element; min : Integer = -1 | let aux : Integer = e.oclAsType(Goal).NDG() in if min = -1 then aux else min.min(aux) endif)
Name	**MaxNDG** – **Max**imum **N**umber of **D**ecompositions of an actor's **G**oal
Informal definition	Maximum number of decompositions associated with a goal that is inside an actor's boundary in the SR model

Table 2. (*Continued.*)

Formal definition	context Actor def:MaxNDG():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Goal) and e.oclAsType(Goal).NDG() > 0) -> iterate(e : Element; max : Integer = -1 | let aux : Integer = e.oclAsType(Goal).NDG() in if max = -1 then aux else max.max(aux) endif)
Name	**AvgNDG** – **A**verage **N**umber of **D**ecompositions of an actor's **G**oal
Informal definition	Average number of decompositions associated with a goal that is inside an actor's boundary in the SR model
Formal definition	context Actor pre:self.NGWDI() > 0 context Actor def:AvgNDG():Real = self.NDG() / self.NGWDI()
Requires	**NDG** – **N**umber of **D**ecompositions of an actor's **G**oal **NGWDI** – **N**umber of **G**oals **W**ith **D**ecompositions **I**nside (see companion site)

Q4 – How complex is a softgoal, with respect to its decomposition?	
Name	**NDS** – **N**umber of **D**ecompositions of an actor's **S**oftgoal
Informal definition	Number of decompositions associated with a softgoal in the SR model
Formal definition	context Softgoal def:NDS():Integer = self.softgoalContribution -> select(cl : ContributionLink | cl.oclIsKindOf(ContributionLink)) -> size()
Name	**MinNDS** – **Min**imum **N**umber of **D**ecompositions of an actor's **S**oftgoal
Informal definition	Minimum number of decompositions associated with a softgoal that is inside an actor's boundary in the SR model
Formal definition	context Actor def:MinNDS():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Softgoal) and e.oclAsType(Softgoal).NDS() > 0) -> iterate(e : Element; min : Integer = -1 | let aux : Integer = e.oclAsType(Softgoal).NDS() in if min = -1 then aux else min.min(aux) endif)
Name	**MaxNDS** – **Max**imum **N**umber of **D**ecompositions of an actor's **S**oftgoal
Informal definition	Maximum number of decompositions associated with a softgoal that is inside an actor's boundary in the SR model
Formal definition	context Actor def:MaxNDS():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Softgoal) and e.oclAsType(Softgoal).NDS() > 0) -> iterate(e : Element; max : Integer = -1 | let aux : Integer = e.oclAsType(Softgoal).NDS() in if max = -1 then aux else max.max(aux) endif)
Name	**AvgNDS** – **A**verage **N**umber of **D**ecompositions of an actor's **S**oftgoal
Informal definition	Average number of decompositions associated with a softgoal that is inside an actor's boundary in the SR model
Formal definition	context Actor pre:self.NSWDI() > 0 context Actor def:AvgNDS():Real = self.NDS() / self.NSWDI()
Requires	**NDS** – **N**umber of **D**ecompositions of an actor's **S**oftgoal **NSWDI** – **N**umber of **S**oftgoals **W**ith **D**ecompositions **I**nside (see companion site)

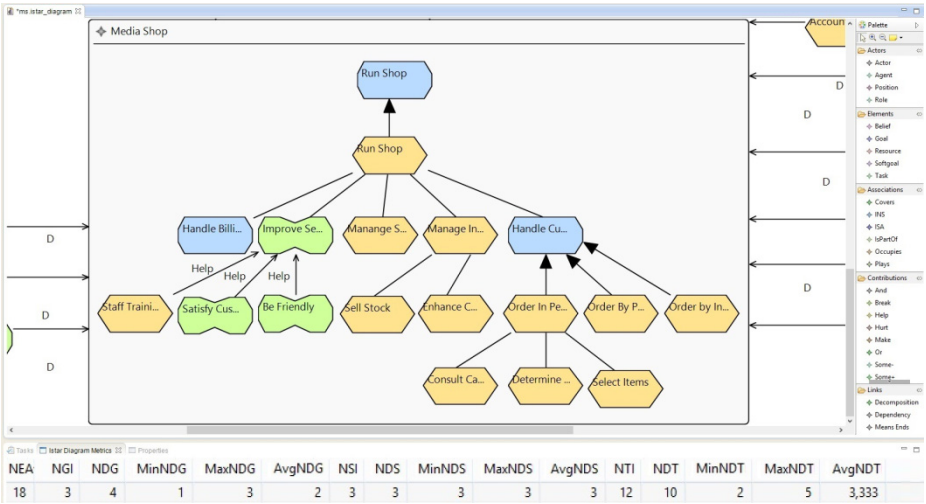
Table 2. (*Continued.*)

Q5 – How complex is a task, with respect to its decompositions?	
Name	**NDT** – **N**umber of **D**ecompositions of an actor's **T**ask
Informal definition	Number of decompositions associated with a task in the SR model
Formal definition	context Task def:NDT():Integer = self.taskDecompositionLink -> select(dl : DecompositionLink | dl.oclIsKindOf(DecompositionLink)) -> size()
Name	**MinNDT** – **Min**imum **N**umber of **D**ecompositions of an actor's **T**ask
Informal definition	Minimum number of decompositions associated with a task that is inside an actor's boundary in the model
Formal definition	context Actor def:MinNDT():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Task) and e.oclAsType(Task).NDT() > 0) -> iterate(e : Element; min : Integer = -1 | let aux : Integer = e.oclAsType(Task).NDT() in if min = -1 then aux else min.min(aux) endif)
Name	**MaxNDT** – **Max**imum **N**umber of **D**ecompositions of an actor's **T**ask
Informal definition	Maximum number of decompositions associated with a task that is inside an actor's boundary in the SR model
Formal definition	context Actor Def:MaxNDT():Integer = self.hasElement -> select(e : Element | e.oclIsKindOf(Task) and e.oclAsType(Task).NDT() > 0) -> iterate(e : Element; max : Integer = -1 | let aux : Integer = e.oclAsType(Task).NDT() in if max = -1 then aux else max.max(aux) endif)
Name	**AvgNDT** – **Av**era**g**e **N**umber of **D**ecompositions of an actor's **T**ask
Informal definition	Average number of decompositions associated with a task that is inside an actor's boundary in the SR model
Formal definition	context Actor pre:self.NTWDI() > 0 context Actor def:AvgNDTI():Real = self.NDT() / self.NTWDI()
Requires	**NDT** – **N**umber of **D**ecompositions of an actor's **T**ask **NTWDI** – **N**umber of **T**asks **W**ith **D**ecompositions **I**nside (see companion site)

3.2 Example

Figure 2 shows a fragment of the Media Shop (MS) case study, whose main objective is to allow an online customer to examine the items in the Medi@ internet catalogue (books, newspapers, magazines, audio CD, videotapes, and the like) and place orders. The figure, taken from our tool, shows the actor Media Shop and some of its elements, as well as the model metrics.

The tool allows to create *i** models using a visual language and provides metrics values for the model. These values can be updated at any time of the construction process. As such, they can be valuable to detect potential problems early in the process, such as a high accidental complexity caused by a modelling option.



NEA	NGI	NDG	MinNDG	MaxNDG	AvgNDG	NSI	NDS	MinNDS	MaxNDS	AvgNDS	NTI	NDT	MinNDT	MaxNDT	AvgNDT
18	3	4	1	3	2	3	3	3	3	3	12	10	2	5	3,333

Fig. 2. Application of the tool and the metrics to the Media Shop case study

4 Evaluation

4.1 Case Studies

To evaluate the presented metrics, we modelled *i** case studies, namely Media Shop (MS) [15], Newspaper Office (NO) [16], Health Care (HC) [2], Health Protection Agency (HPA) [17] and National Air Traffic Services (NATS) [17] with our tool, and then collected the corresponding metrics. The case studies MS, NO, and HC have been extensively used in the literature, while HPA and NATS are real-world case studies, also available in the literature. They target different domains and have different essential complexities. A common characteristic of these models is that they are available with full details, making them good candidates for evaluation.

4.2 Results and Discussion

In this section we present the main findings from our statistics analysis of the collected metrics. The statistics data files and scripts for performing the statistics analysis outlined here can be found in the paper's companion site.

Concerning model size (***Q1,*** Fig. 3a and Fig. 3b), the NATS (National Air Traffic Services) system has, approximately, twice the size of the second largest system (HPA, Health Protection Agency). The HC (Health Care) system has less actors, but more elements than the MS (Media Shop) and NO (Newspaper Office) systems, which have a very similar size. In fact, if we compute the elements to actors ratio (Fig. 3c), we note that HC has a higher ratio than all the other systems, which have very similar ratios. This may suggest that HC could be an interesting candidate for

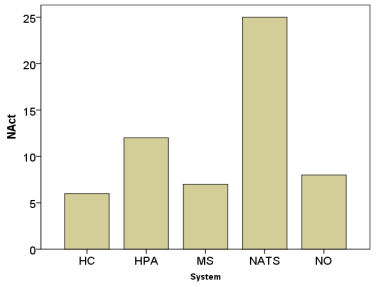
refactoring. In contrast, we note that the most complex system, in terms of size, has the lowest element/actor density, suggesting a good overall modularity.

This overview on complexity is but a first impression. We need to analyze more detailed features to get a clearer picture of the modularity profile of these systems. For each of the counting metrics NDG, NDS and NDT (number of decompositions of an actor's goal, softgoal and task, respectively), we present here a boxplot chart with their distributions on the actors of their corresponding systems, where we can identify the outliers (denoted with O) and extremes (denoted with *), in Fig. 3d-f.

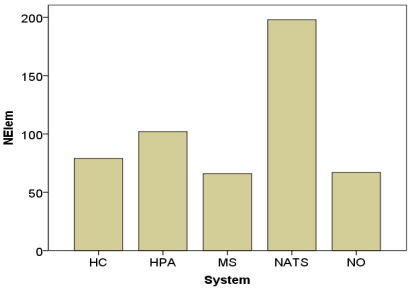
A closer inspection on the boxplot graphs (Fig. 3d-f) shows that there are two actors which present outlier, or even extreme values, in NDS and NDT. These should be our most likely candidates for further scrutiny. For example, the actor *Civil ATCO* (Civilian Air Traffic Controller), from the National Air Traffic Services system, has an outlier value for the softgoal decomposition (NDS) and an extreme value for the task decomposition (NDT) metrics. *Civil ATCO* is a crucial actor in that system, whose specification is much more complex than that of most other actors in the same system.

There are at least two possible problems that should be checked, concerning the *Civil ATCO* actor's decomposition. A first potential problem is that this actor may have too many responsibilities. A typical refactoring would be to decompose the actor into *sub-actors*, using the *is-part-of* relationship, where each *sub-actor* would be responsible for a *sub-system*. This anti-pattern and its proposed refactoring are similar to god classes [18] and their refactoring, in object-oriented design. Note that, sometimes, the extra complexity is not of an accidental nature, but rather of an essential one. In such case, this analysis is still useful, in the sense that it highlights an actor in the system which has an extremely high essential complexity associated with it. This may hint project managers to assign more resources to quality assurance activities (e.g. inspections and testing) to artifacts related to the implementation of the requirements associated with this actor.

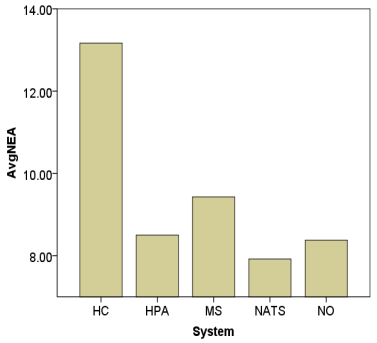
It may also be the case that the requirements engineer may over-decompose these goals, softgoals, or tasks, by following a functional decomposition strategy, leading to poor modularity. This is similar to the *functional decomposition* anti-pattern [18], where the encapsulation principle is neglected. Another consequence is that the abstraction level of the model lowers: including too many (design) details may obfuscate the requirements model, making it harder to understand and evolve. Abstracting away the unnecessary detailed decompositions can improve the overall modularity of the requirements model.



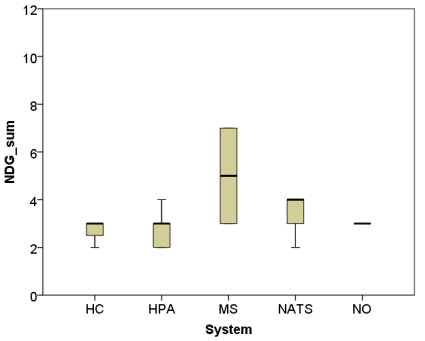
a) Number of actors in the system



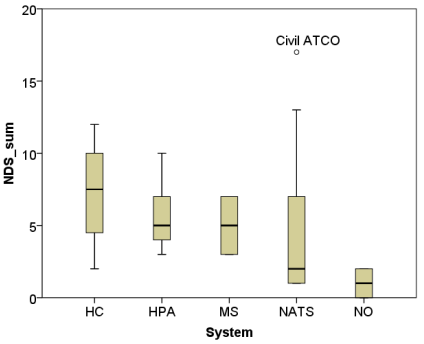
b) Number of elements in the system



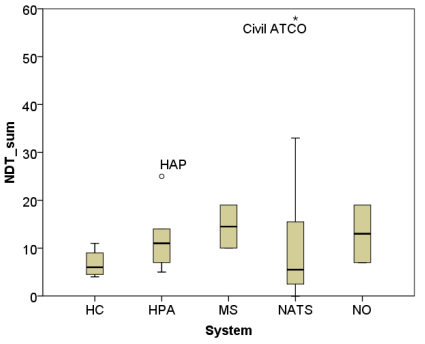
c) Goal decompositions per actor



d) NDG distribution



e) NDS distribution



f) NDT distribution

Fig. 3. Metrics values for our case studies

5 Related Work

Horkoff and Yu [19] evaluate seven goal satisfaction analysis procedures using available tools that implement those procedures. They evaluate three sample goal models. The results help to understand the ways in which procedural design choices affect

analysis results, and how differences in analysis results could lead to different recommendations over alternatives in the model. Compared to our work, they study a different aspect of goal modelling, i.e. goal satisfaction analysis, not complexity.

Hilts and Yu [20] describe the Goal-Oriented Design Knowledge Library (GO-DKL) framework. This framework provides an approach for extracting, coding and storing relational excerpts of design knowledge from academic publications. This framework was designed for knowledge reuse purposes. Our work could extend that framework by providing information about the complexity of those existing models.

Ramos et al. [21] 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. 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. [22] 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 [23] 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 [24]. The method is applied to evaluate business process performance. Their approach is more focused on the process, and more generic, while we focus on modularity assessment of *i** models.

6 Conclusions

In this paper, we proposed a metrics suite for evaluating the complexity of *i** goal models, formally specified (using OCL), implemented and incorporated in an eclipse based modelling tool. The metrics were proposed using the GQM approach. The selected questions allow evaluating the complexity of the model as a whole concerning its size, and the complexity of an actor and its model elements (i.e. goals, softgoals and tasks). The set of metrics provide quantitative information to answer the corresponding questions. The contribution of this paper is that evaluating complexity at early stages to identify modularity problems of the models allows avoiding eventual

extra costs in during the later stages of software development and also during software maintenance and evolution. The realization that the modularity of a requirements model can be improved can trigger requirements refactoring opportunities, like decomposing a system's actor using an is-part-of relationship between sub-actors, or abstracting over-decomposed goals, softgoals, or tasks. These metrics were validated by applying them to well-known industrial and academic case studies. The results of these metrics also reveal a pattern of usage in goal modelling concerning modularity of those models.

For future work, we intend to replicate this evaluation with other *i** models and extend the metrics set to cover other model quality attributes, such as correctness. The final aim is to provide a metrics-based modelling support in GORE tools. In particular, with an increased number of evaluated models, we will be able to, for example, identify thresholds for suggesting merging and/or decomposing model elements to reduce complexity of an *i** model. As a cross-validation for those thresholds, we plan to conduct an experiment with requirements engineers to assess the extent to which those thresholds are correlated with an increased difficulty in *i** model comprehension. We also plan to define and apply refactoring patterns for GORE models.

Acknowledgments. The authors would like to thank FCT/UNL and CITI – PEst-OE/EEI/UI0527/2011, for the financial support for this work.

References

1. Van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Proceedings Fifth IEEE International Symposium on Requirements Engineering, pp. 249–262. IEEE Comput. Soc. (2001)
2. Yu, E.: Modelling Strategic Relationships for Process Reengineering, PhD dissertation, University of Toronto, Canada (1995)
3. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley (2009)
4. ITU-T: Recommendation Z.151 (10/12): User Requirements Notation (URN)–Language definition. , Geneva, Switzerland (2012)
5. Brooks, F.P.: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley Publishing Company, Reading (1995)
6. ISO/IEC JTC1, O.M.G.: Information technology - Object Management Group Object Constraint Language (OCL), ISO/IEC 19507 (2012)
7. Espada, P., Goulão, M., Araújo, J.: A Framework to Evaluate Complexity and Completeness of KAOS Goal Models. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 562–577. Springer, Heidelberg (2013)
8. Almeida, C., Goulão, M., Araújo, J.: A Systematic Comparison of i * Modelling Tools Based on Syntactic and Well-formedness Rules. In: Castro, J., Horkoff, J., Maiden, N., Yu, E. (eds.) 6th International i* Workshop (iStar 2013). CEUR, vol. 978, pp. 43–48. CEUR Workshop Proceedings (2013)
9. i* wiki, `http://istarwiki.org/` (last access: March 2014)
10. Kolovos, D., Rose, L., García-Domínguez, A., Paige, R.: The Epsilon Book. Eclipse Foundation (2013)

11. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2009)
12. Eclipse: GMF, http://www.eclipse.org/modeling/gmp/?project=gmf-tooling (last access: March 2014)
13. Eclipse: Ecore tools, http://wiki.eclipse.org/index.php/Ecore_Tools (last access: March 2014)
14. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. Encyclopedia of Software Engineering 2, 528–532 (1994)
15. Castro, J., Kolp, M., Mylopoulos, J.: A Requirements-Driven Development Methodology. In: Dittrich, K.R., Geppert, A., Norrie, M. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 108–123. Springer, Heidelberg (2001)
16. Silva, C., Castro, J., Tedesco, P., Silva, I.: Describing Agent-Oriented Design Patterns in Tropos. In: Brazilian Symposium on Software Engineering (SBES 2005), pp. 10–25 (2005)
17. Lockerbie, J., Maiden, N.A.M., Engmann, J., Randall, D., Jones, S., Bush, D.: Exploring the impact of software requirements on system-wide goals: a method using satisfaction arguments and i* goal modelling. Requir. Eng. 17, 227–254 (2011)
18. Brown, W.J., Malveau, R.C., McCormick, H.W., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley (1998)
19. Horkoff, J., Yu, E.: Comparison and evaluation of goal-oriented satisfaction analysis techniques. Requir. Eng. 18, 199–222 (2012)
20. Hilts, A., Yu, E.: Design and evaluation of the goal-oriented design knowledge library framework. In: Proc. 2012 iConference (iConference 2012), pp. 384–391 (2012)
21. Ramos, R., Castro, J., Araújo, J., Moreira, A., Alencar, F., Santos, E., Penteado, R., Carlos, S., Paulo, S.: AIRDoc – An Approach to Improve Requirements Documents. In: Brazilian Symposium on Software Engineering, SBES 2008 (2008)
22. De Vasconcelos, A.M.L., de la Vara, J.L., Sanchez, J., Pastor, O.: Towards CMMI-compliant Business Process-Driven Requirements Engineering. In: Eighth Int. Conf. Qual. Inf. Commun. Technol (QUATIC 2012), pp.193–198 (2012)
23. 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)
24. 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)