Metrics for Measuring Complexity and Completeness for Social Goal Models

Catarina Gralha, João Araújo, Miguel Goulão

NOVA-LINCS, Departamento de Informática,
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Abstract

Goal-oriented Requirements Engineering approaches have become popular in the Requirements Engineering community as they provide expressive modelling languages for requirements elicitation and analysis. However, as a common challenge, such approaches are still struggling when it comes to managing the accidental complexity of their models. Furthermore, those models might be incomplete, resulting in insufficient information for proper understanding and implementation. In this paper, we provide a set of metrics, which are formally specified and have tool support, to measure and analyse complexity and completeness of goal models, in particular social goal models (e.g. *i**). Concerning complexity, the aim is to identify refactoring opportunities to improve the modularity of those models, and consequently reduce their accidental complexity. With respect to completeness, the goal is to automatically detect model incompleteness. We evaluate these metrics by applying them to a set of well-known system models from industry and academia. Our results suggest refactoring opportunities in the evaluated models, and provide a timely feedback mechanism for requirements engineers on how close they are to completing their 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, organising, and structuring requirements, as well as in exploring and evaluating alternative solutions to a problem [1]. There are two types of GORE models: models that only capture goals and their refinements (e.g. KAOS [2]), or models that capture the actors behind the goals and the way they deal with them through refinement and delegation (e.g. . *i** [3] and GRL [4]). Earlier work (by the authors) focused on the KAOS goal model [5]. In this work, we motivate and propose metrics for social goal models, such as those used in *i**.

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 minimise, as much as possible, the accidental complexity introduced by the way we model those problems [6].

A possible way of minimising 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. Collecting such metrics on several different models is a necessary step to establish a typical usage profile of the modelling mechanisms.

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 modeller for a refactoring opportunity where this agent should in fact be decomposed into several sub-agents.

Another challenge for the GORE approaches is that resulting models might be incomplete, which results in lack of information for its proper understanding and implementation. Eliciting requirements for these systems

Email address: acg.almeida@campus.fct.unl.pt, joao.araujo@fct.unl.pt, mgoul@fct.unl.pt (Catarina Gralha, João Araújo, Miguel Goulão)

is typically performed in a stepwise manner. The requirements engineer begins by modelling the higher-level elements, and then decompose them into less abstract elements. In this refinement process, it is useful to have a measure of completeness and a timely feedback mechanism, which can help requirements engineers to realise how close they are to completing their models. In this paper, we focus on how we can automatically detect *i** models incompleteness. We define a metrics suite for assessing their completeness and the completeness of the elements defined in them.

The objective of this paper is to provide a metrics suite, along with corresponding tool support, targeted to the measurement and analysis of complexity and completeness of social goal models (in this paper, *i** models). The goal is the identification of refactoring opportunities to improve the modularity of those models, and automatically detect social goal models incompleteness. The identification of such opportunities can be useful during the development of the system, where a better modularisation 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. Regarding models' completeness, measuring the current status of a model, and its level of completeness at a given time, can help in calculating the estimated effort required to finish the modelling process.

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) [7] upon the *i** meta-model. 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 [5], 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 social goal models. While KAOS builds on concepts such as goals and refinements, the metrics proposed in this paper are specific to a set of modelling constructs, e.g. actors, goals, refinements, and delegations, commonly present in social goal models. 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 extends our previous work in [8] by enhancing the initial set of complexity metrics and adding a full set of completeness metrics.

The rest of the paper is organised 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 models, the results obtained by applying the metrics on those models, 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** meta-model, and the detailed specification of auxiliary metrics can be found in this paper's companion site (CS) [1].

2. The *i** approach

The *i** [3] approach was developed for modelling and reasoning about organisational environments and their information systems. It focuses on the concept of intentional actor. Actors in their organisational 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 organisational 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, resources and beliefs) are related by means-end or decomposition

[1] https://sites.google.com/site/miguelgoulaofct/is2015companion

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 sub-goal, a sub-task, 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 and the completeness 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 provide adequate support for a flexible definition of such metrics (detailed comparison of the existing *i** tool support can be found in [9, 10]). 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 [11], EMF/GMF [12, 13] and Ecore Tools [14].

Figure 1 presents a fragment of the *i** meta-model implemented in our tool, showing only the concepts which will be used in the metrics definitions proposed in this paper. This meta-model is the basis for the tool support for the specification of *i** models, and their evaluation with model metrics. The root of the meta-model is the meta-class ISTAR, which contains all the nodes and relationships of an *i** model. This top-level meta-class serves as a basis for model analysis. The remaining meta-classes can be easily mapped into some of the concepts described earlier in this section.

3. A metrics set for *i**

We propose a metrics-based analysis framework for *i** models, using the Goal-Question-Metric (GQM) approach [15].

Table 1 summarises the GQM-based proposal for a set of metrics that will allow satisfying the goal of complexity and completeness evaluation. The first column (*Question*) presents questions that will allow evaluating whether the overall goal is being achieved. The second column (*Metric*) shows a set of metrics that provide quantitative information to answer the corresponding question. The complexity evaluation goal is related with the model and its elements. Question Q1 concerns complexity, as perceived when regarding the model as a whole. In particular, we are interested in the number of actors and in the number of elements, within a given model. The next set of questions are targeted to assessing the complexity of model elements, namely the amount of responsibilities supported by an actor in the model (Q2), and the number of decompositions of actor's goals (Q3), softgoals (Q4) and tasks (Q5). For each of these elements-centred questions, we define a basic metric (e.g. NEA, for question Q2) and three additional distribution metrics presenting the minimum, maximum and average values for the basic metric. Questions Q6 and Q7 quantify the dependency relationships of an actor, and we are interested in the percentage of outgoing (Q6) and incoming dependencies (Q7) of such actor. Lastly, Q8 allows to infer if the complexity of a certain actor is related with its type (that is, actor, agent, position or role).

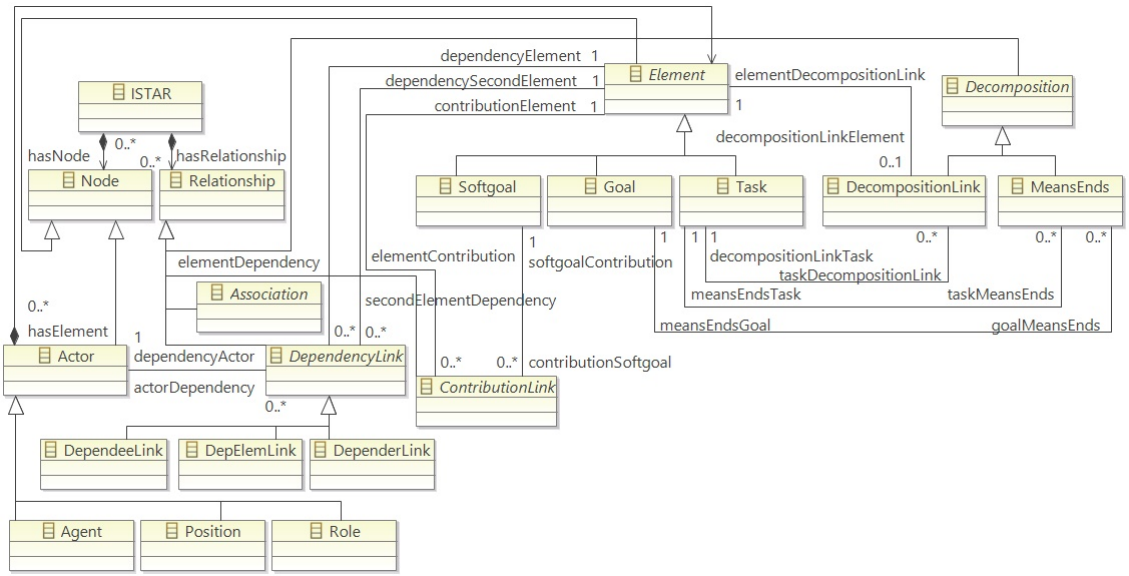


Figure 1: Partial *i** meta-model

Table 1: Goal-Question-Metric for *i** models complexity and completeness 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**era**g**e **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 a **G**oal **MinNDG** – **Min**imum **N**umber of **D**ecompositions of a **G**oal **MaxNDG** – **Max**imum **N**umber of **D**ecompositions of a **G**oal **AvgNDG** – **Av**era**g**e **N**umber of **D**ecompositions of a **G**oal
Q4 – How complex is an actor's softgoal, with respect to its decomposition?	**NDS** – **N**umber of **D**ecompositions of a **S**oftgoal **MinNDS** – **Min**imum **N**umber of **D**ecompositions of a **S**oftgoal **MaxNDS** – **Max**imum **N**umber of **D**ecompositions of a **S**oftgoal **AvgNDS** – **Av**era**g**e **N**umber of **D**ecompositions of a **S**oftgoal
Q5 – How complex is an actor's task, with respect to its decompositions?	**NDT** – **N**umber of **D**ecompositions of a **T**ask **MinNDT** – **Min**imum **N**umber of **D**ecompositions of a **T**ask **MaxNDT** – **Max**imum **N**umber of **D**ecompositions of a **T**ask **AvgNDT** – **Av**era**g**e **N**umber of **D**ecompositions of a **T**ask
Q6 – Is an actor too dependent in the model?	**POD** – **P**ercentage of **O**utgoing **D**ependencies
Q7 – Does an actor have too many dependencies in the model?	**PID** – **P**ercentage of **I**ncoming **D**ependencies
Q8 – Is there a variation in the average complexity of the different types of actors?	**NEIAct** – **N**umber of **E**lements **I**nside an **A**ctor **NEIAg** – **N**umber of **E**lements **I**nside an **A**gent **NEIP** – **N**umber of **E**lements **I**nside a **P**osition **NEIR** – **N**umber of **E**lements **I**nside a **R**ole

Goal: Completeness evaluation	
Question	**Metric**
Q9 – How specific are the actors?	**PSAct** – **P**ercentage of **S**pecific **A**ctors
Q10 – How detailed are the goals?	**PGWME** – **P**ercentage of **G**oals **W**ith **M**eans-**e**nd
Q11 – How detailed are the softgoals?	**PSWC** – **P**ercentage of **S**oftgoals **W**ith **C**ontributions
Q12 – How detailed is the SR model with respect to its actors?	**PAWEI** – **P**ercentage of **A**ctors **W**ith **E**lements **I**nside
Q13 – How close are we to end the assignment of responsibilities to an actor?	**PAWOUEI** – **P**ercentage of **A**ctors **W**ith**O**ut **U**nconnected **E**lements **I**nside
Q14 – How close are we to end the assignment of links to the actors?	**PAWDOA** – **P**ercentage of **A**ctors **W**ith **D**ependencies **O**r **A**ssociations

The completeness evaluation goal is related with the requirements that were identified in interactions with stakeholders. The first questions are targeted to assessing the detail of actors' specification, and the detail level of goals and softgoals. In particular, we are interested in the percentage of actors with a specific type (Q9), goals with means-ends (Q10) and softgoals with contribution links (Q11). Question Q12 quantifies the percentage of actors with elements inside its boundary. Finally, the last questions allows to assess how complete is the model and how close we are to finish the modelling process. In particular, we are interested in the assignment of responsibilities to an actor (Q13) and in the assignment of links (namely dependencies and associations) to and between actors (Q14).

3.1. Complexity metrics definition

In this section we present the complexity metrics outlined in section 3. For each question we provide a table containing information concerning (i) the symptom the requirements engineer should be alert to (in terms of detecting *"unusual values"*, when compared to other projects - more precisely, outlier and extreme values); (ii) the potential problem this symptom may indicate (note that *"suspicious"* metrics values do not necessarily imply that there is a problem - they just suggest it may be worth checking if there is one, thus helping in early problem identification and mitigation); (iii) a suggested action that the requirements engineer may want to take, if after inspecting the corresponding model elements (s)he decides there is an actual problem worth fixing; (iv) an informal definition of the metrics specified to answer it; and (v) a formal definition using OCL upon the meta-model fragment presented in figure 1. When required, we also 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, softgoals, or tasks, to be decomposed. Elements without decompositions may have been modelled in order to be final elements. It would not make sense analysing the extent to which they are decomposed. For the sake of brevity, we omit trivial auxiliary metrics definitions with basic counts. The full metrics suite definition in OCL, including all auxiliary metrics, can be found in the paper's companion site.

Regarding question Q1 (see table 2), 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.

Table 2: **Q1** – How complex is the model, concerning the number of actors and elements?

Symptom	The size of the model is unusually lower, or higher, than in most models.
Potential problem	The model may be over-simplistic, or unnecessarily complex, leading to problems in the understandability of the system.
Suggested action	Consider revising the model. If necessary, add more detail, or remove accidental complexity.
Metric	**NAct** – **N**umber of **Act**ors
Informal definition	Total number of actors in the SD/SR model

Formal definition	`context ISTAR`
	`def:NAct():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) -> size()`

Metric	**NElem** – **N**umber of **Elem**ents
Informal definition	Total 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 (CS) **NEIAB** – **N**umber of **E**lements **I**nside **A**ctors' **B**oundaries (CS)

Table 3: **Q2** – Does an actor have too many responsibilities in the model?

Symptom	The actor has an unusually high number of internal model elements.
Potential problem	The actor may have too many responsibilities in the model.
Suggested action	This actor may be a good candidate for further scrutiny. Consider decomposing this actor into several sub-actors and distributing his responsibilities among them. If the system has no outliers, the assignment of responsibilities is probably well balanced.
Metric	**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()`

Metric	**MinNEA** – **Min**imum **N**umber of **E**lements of an **A**ctor
Informal definition	Minimum number of elements inside an actor's boundary

Formal definition	`context ISTAR`
	`def:MinNEA():Integer  = self.hasNode ->`
	`select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; min:Integer = -1 |`
	`  let nea:Integer = n.oclAsType(Actor).NEA() in`
	`  if min = -1 then nea else min.min(nea) endif)`

Metric	**MaxNEA** – **Max**imum **N**umber of **E**lements of an **A**ctor
Informal definition	Maximum number of elements inside an actor's boundary

Formal definition	`context ISTAR`
	`def:MaxNEA():Integer = self.hasNode ->`
	`  select(n:Node | n.oclIsKindOf(Actor)) ->`
	`  iterate(n:Node; max:Integer = -1 |`
	`  let nea:Integer = n.oclAsType(Actor).NEA() in`
	`  if max = -1 then aux else max.max(nea) endif)`

Metric	**AvgNEA** – **A**verage **N**umber of **E**lements of an **A**ctor
Informal definition	Average number of elements inside an actor's boundary

Formal definition	`context ISTAR::AvgNEA()`
	`pre:self.NAct() > 0`
	
	`context ISTAR`
	`def:AvgNEA():Double = self.NEA() / self.NAct()`

Requires	**NEA** – **N**umber of **E**lements of an **A**ctor **NAct** – **N**umber of **Act**ors

Table 4: **Q3** – How complex is an actor's goal, with respect to its decompositions?

Symptom	An actor's goal has an unusually high number of decompositions.
Potential problem	The goal may be over-decomposed.
Suggested action	This goal may be a good candidate for further scrutiny. Consider abstracting out this goal, if it is over-decomposed. If the actor has no outlier goals, their decomposition is probably well balanced.
Metric	**NDG** – **N**umber of **D**ecompositions of a **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()`

Metric	**MinNDG** – **Minimum N**umber of **D**ecompositions of a **G**oal
Informal definition	Minimum number of decompositions associated with a goal

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 ndg:Integer = e.oclAsType(Goal).NDG() in`
	`if min = -1 then ndg else min.min(ndg) endif)`

Metric	**MaxNDG** – **Maximum N**umber of **D**ecompositions of a **G**oal
Informal definition	Maximum number of decompositions associated with a goal

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 ndg:Integer = e.oclAsType(Goal).NDG() in`
	`if max = -1 then ndg else max.max(ndg) endif)`

Metric	**AvgNDG** – **Avera**ge **N**umber of **D**ecompositions of a **G**oal
Informal definition	Average number of decompositions associated with a goal

Formal definition	`context Actor::AvgNDG()`
	`pre:self.NGWDI() > 0`
	
	`context Actor`
	`def:AvgNDG():Double = self.NDG() / self.NGWDI()`

Requires	**NDG** – **N**umber of **D**ecompositions of a **G**oal **NGWDI** – **N**umber of **G**oals **W**ith **D**ecompositions **I**nside (CS)

Different candidate models for the same system can be compared, using these metrics, with respect to their overall complexity. Work on other paradigms (e.g. Object-Oriented) has collected empirical evidence on the positive correlation between size and complexity [16, 17]. Over-simplistic models may be insufficiently detailed, leading to problems in its understandability. On the other hand, if the system is unnecessarily complex, understandability problems may also occur.

Concerning question Q2 (see table 3), a high value for NEA (number of elements of an actor) can be an indicator that a particular actor has too much responsibility in the model, being harder to manage the responsibilities in an efficient manner. The minimum, maximum and average values help the requirements engineer recognising cases where the responsibility is higher or lower than expected. Complexity can also be used for supporting project estimation efforts.

Questions Q3, Q4 and Q5, (see tables 4, 5 and 6, respectively) 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. It is important to note that including too many design details in the model, by over-decomposing goals, softgoals and tasks, may obfuscate the requirements model, making it harder to understand and evolve.

Concerning questions Q6 and Q7 (see tables 7 and 8,

Table 5: **Q4** – How complex in an actor's softgoal, with respect to its decompositions?

Symptom	An actor's softgoal has an unusually high number of decompositions.
Potential problem	The softgoal may be over-decomposed.
Suggested action	This softgoal may be a good candidate for further scrutiny. Consider abstracting out this softgoal, if it is over-decomposed. If the actor has no outlier softgoals, their decomposition is probably well balanced.
Metric	**NDS** – **N**umber of **D**ecompositions of a **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()`

Metric	**MinNDS** – **Min**imum **N**umber of **D**ecompositions of a **S**oftgoal
Informal definition	Minimum number of decompositions associated with a softgoal

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 nds:Integer = e.oclAsType(Softgoal).NDS() in`
	`if min = -1 then nds else min.min(nds) endif)`

Metric	**MaxNDS** – **Max**imum **N**umber of **D**ecompositions of a **S**oftgoal
Informal definition	Maximum number of decompositions associated with a softgoal

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 nds:Integer = e.oclAsType(Softgoal).NDS() in`
	`if max = -1 then nds else max.max(nds) endif)`

Metric	**AvgNDS** – **Av**era**g**e **N**umber of **D**ecompositions of a **S**oftgoal
Informal definition	Average number of decompositions associated with a softgoal

Formal definition	`context Actor::AvgNDS()`
	`pre:self.NSWDI() > 0`
	
	`context Actor`
	`def:AvgNDS():Double = self.NDS() / self.NSWDI()`

Requires	**NDS** – **N**umber of **D**ecompositions of a **S**oftgoal **NSWDI** – **N**umber of **S**oftgoals **W**ith **D**ecompositions **I**nside (CS)

respectively), the values of POD (percentage of outgoing dependencies) and PID (percentage of incoming dependencies) are measures of an actor's dependency links in the SD/SR model. These values can be used to verify if an actor is a source or a sink, allowing the identification of pathological situations and actors' archetypes in the system. Furthermore, too many dependencies increase the complexity and reduces the encapsulation and reuse potential of the actors. Excessive dependencies also limit the understandability and maintainability of the system. In addition, if too many actors depend on a particular actor, changes in this actor may have ripple effects through the other actors, potentially reducing the maintainability of the system.

Question Q8 (see table 9) provide information about the complexity of generic and specific actors, that is: actors, agents, positions and roles. The value of NEIAct (number of elements inside an actor), NEIAg (number of elements inside an agent), NEIP (number of elements inside a position) and NEIR (number of elements inside a role) allows to verify if the complexity of a certain actor is related with its type. It might be the case that a particular type of actor is frequently over- or under-specified, which would reflect on the typical complexity of actors of that type. Note that excessive use of the specific actor notations might lead to more complex models that might become harder to deal with and understand.

3.2. Completeness metrics definition

In this section we present the completeness metrics outlined in section 3. The tables structure is similar to the one presented in section 3.1.

Regarding question Q9 (see table 10), the value of PSAct (percentage of specific actors) is a measure of the actors' specification. The usage of specialised actor notations such as agents, roles, and positions, when the distinction between them is easily made, can help in gaining higher level of detailing in instantiating the stakeholders and capturing the knowledge domain. Lack of use of any of these specialised actor notations might subject the model to lose some useful information. However, excessive use of the special actor no-

Table 6: **Q5** – How complex in an actor's task, with respect to its decompositions?

Symptom	An actor's task has an unusually high number of decompositions.
Potential problem	The task may be over-decomposed.
Suggested action	This task may be a good candidate for further scrutiny. Consider abstracting out this task, if it is over-decomposed. If the actor has no outlier tasks, their decomposition is probably well balanced.
Metric	**NDT** – **N**umber of **D**ecompositions of a **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()`

Metric	**MinNDT** – **Min**imum **N**umber of **D**ecompositions of a **T**ask
Informal definition	Minimum number of decompositions associated with a task

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 ndt : Integer = e.oclAsType(Task).NDT() in`
	`if min = -1 then ndt else min.min(ndt) endif)`

Metric	**MaxNDT** – **Max**imum **N**umber of **D**ecompositions of a **T**ask
Informal definition	Maximum number of decompositions associated with a task

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 ndt:Integer = e.oclAsType(Task).NDT() in`
	`if max = -1 then ndt else max.max(ndt) endif)`

Metric	**AvgNDT** – **Av**erage **N**umber of **D**ecompositions of a **T**ask
Informal definition	Average number of decompositions associated with a task

Formal definition	`context Actor::AvgNDTI()`
	`pre:self.NTWDI() > 0`
	
	`context Actor`
	`def:AvgNDTI():Double = self.NDT() / self.NTWDI()`

Requires	**NDT** – **N**umber of **D**ecompositions of a **T**ask **NTWDI** – **N**umber of **T**asks **W**ith **D**ecompositions **I**nside (CS)

Table 7: **Q6** – Is an actor too dependent in the model?

Symptom	An actor has an unusually high number of outgoing dependencies.
Potential problem	The actor may be too dependent on other actors to achieve its goals.
Suggested action	This actor may be a good candidate for further scrutiny. Consider balancing the number of outgoing and incoming dependencies among actors. If there are no outliers, the dependencies are probably well balanced.
Metric	**POD** – **P**ercentage of **O**utgoing **D**ependencies
Informal definition	Percentage of outgoing dependencies of an actor in the SD/SR model

Formal definition	`context Actor::POD()`
	`pre: self.ND() > 0`
	
	`context Actor`
	`def:POD():Double = self.NOD() / self.ND()`

Requires	**NOD** – **N**umber of **O**utgoing **D**ependencies (CS) **ND** – **N**umber of **D**ependencies (CS)

tations might lead to much more complex models that might become harder to deal with. Therefore, the choice for the use of the general actor versus the specialised actor notation could be made based on the value and additional information that they will add to the model [18]. Nevertheless, this metric is a useful measure that can be used to compare the specification of actors from different models, and assess whether there is any effect by having a higher precision in the specification level of the actors.

Questions Q10 and Q11 (see tables 11 and 12, respectively) provide different perspectives on the completeness associated with a particular actor. The value of PGWME (percentage of an actor's goals with means-

Table 8: **Q7** – Does an actor have too much dependencies in the model?

Symptom	An actor has an unusually high number of incoming dependencies.
Potential problem	If too many actors depend on a particular actor, changes in this actor may have ripple effects through the other actors. This potentially reduces the maintainability of a system.
Suggested action	This actor may be a good candidate for further scrutiny. Consider balancing the number of outgoing and incoming dependencies among actors. If there are no outliers, the dependencies are probably well balanced.
Metric	**PID** – **P**ercentage of **I**ncoming **D**ependencies
Informal definition	Percentage of incoming dependencies of an actor in the SD/SR model

Formal definition	`context Actor::PID()`
	`pre: self.ND() > 0`
	
	`context Actor`
	`def:PID():Double = self.NID() / self.ND()`

Requires	**NID** – **N**umber of **I**ncoming **D**ependencies (CS) **ND** – **N**umber of **D**ependencies (CS)

Table 9: **Q8** – Is there a significant difference in the average complexity of the different types of actors?

Symptom	The actors from different types have a significantly different complexity. This is observable by statistically comparing the distributions of complexity by actor type.
Potential problem	It might be the case that a particular type of actor is frequently over- or under-specified, which would reflect on the typical complexity of actors of that type. On the other hand there might be a good reason for making a particular actor type more (or less) complex than the other ones.
Suggested action	Consider comparing the average complexity of the different actor types with the one found in other systems. If it is significantly different, further investigate whether this results from the essential complexity of the system, or from some accidental factor (such as over-, or under-specification of the actors).
Metric	**NEIAct** – **N**umber of **E**lements **I**nside an **A**ctor
Informal definition	Number of elements inside an actor's boundary in the SR model

Formal definition	`context ISTAR`
	`def:NEIAct():Integer = self.NEIAB() -`
	`(self.NEIAgB() + self.NEIPB() + self.NEIRB())`

Requires	**NEIAB** – **N**umber of **E**lements **I**nside **A**ctors' **B**oundaries (CS) **NEIAgB** – **N**umber of **E**lements **I**nside **Ag**ents' **B**oundaries (CS) **NEIPB** – **N**umber of **E**lements **I**nside **P**ositions' **B**oundaries (CS) **NEIRB** – **N**umber of **E**lements **I**nside **R**oles' **B**oundaries (CS)
Metric	**NEIAg** – **N**umber of **E**lements **I**nside an **A**gent
Informal definition	Number of elements inside an agent's boundary in the SR model

Formal definition	`context Agent`
	`def:NEIAg():Integer = self.hasElement ->`
	`select(e:Element | e.oclIsKindOf(Element)) -> size()`

Metric	**NEIP** – **N**umber of **E**lements **I**nside a **P**osition
Informal definition	Number of elements inside a position's boundary in the SR model

Formal definition	`context Position`
	`def:NEIP():Integer = self.hasElement ->`
	`select(e:Element | e.oclIsKindOf(Element)) -> size()`

Metric	**NEIR** – **N**umber of **E**lements **I**nside a **R**ole
Informal definition	Number of elements inside a role's boundary in the SR model

Formal definition	`context Role`
	`def:NEIR():Integer = self.hasElement ->`
	`select(e:Element | e.oclIsKindOf(Element)) -> size()`

end link), presented in Q10, measures the completeness of goals decompositions associated with an actor. The value of PSWC (percentage of an actor's softgoals with contribution links), presented in Q11, measure the completeness of softgoals decompositions associated with an actor. The higher the value of these metrics, the higher the actor's level of completeness. A low number of decomposition of goals and softgoals might subject the model to lose some useful information, since the level of precision and detail is lower, leading to understandability problems on how goals and softgoals can be achieved.

With respect to question Q12 (see table 13), the value of PAWEI (percentage of actors with elements inside its boundary) provide information about how detailed the SR model is with respect to its actors. If there are actors without elements inside, they may not offer any relevant information. If those actors are useful to the model, it is advisable that they are detailed, ergo, having a higher level of completeness.

Table 10: **Q9** – How specific are the actors?

Symptom	This system uses a significantly different percentage of specific actors, when compared to other systems.
Potential problem	This may be a symptom of an insufficiently detailed system specification, or, conversely, an over-specified one which may be difficult to understand.
Suggested action	Consider scrutinising the types of actors used in the model and re-considering whether an actor should, or should not be defined using a specific type.
Metric	**PSAct** – **P**ercentage of **S**pecific **A**ctors
Informal definition	Percentage of actors with a specific type (agent, position or role)

Formal definition	`context ISTAR::PSAct()`
	`pre: self.NAct() > 0`
	
	`context ISTAR`
	`def:PSAct():Double = (self.NAgents() +`
	`self.NRoles() + self.NPos()) / self.NAct()`

Requires	**NAgents** – **N**umber of **Agents** (CS) **NRoles** – **N**umber of **Roles** (CS) **NPos** – **N**umber of **Pos**itions (CS) **NAct** – **N**umber of **Act**ors

Table 11: **Q10** – How detailed are the goals?

Symptom	An actor's goal has an unusually low number of decompositions.
Potential problem	The goal may be under-decomposed.
Suggested action	This goal may be a good candidate for further scrutiny. Consider detailing this goal, if it is under-decomposed. If the actor has no outlier goals, their detail is probably well balanced.
Metric	**PGWME** – **P**ercentage of **G**oals **W**ith **M**eans-**e**nd
Informal definition	Percentage of goals with means-end link

Formal definition	`context ISTAR::PGWME()`
	`pre: self.NGIAB() > 0`
	
	`context ISTAR`
	`def:PGWME():Double = self.NGWD() / self.NGIAB()`

Requires	**NGWD** – **N**umber of **G**oals **W**ith **D**ecompositions (CS) **NGIAB** – **N**umber of **G**oals **I**nside **A**ctors' **B**oundaries (CS)

Table 12: **Q11** – How detailed are the softgoals?

Symptom	An actor's softgoal has an unusually low number of decompositions.
Potential problem	The softgoal may be under-decomposed.
Suggested action	This softgoal may be a good candidate for further scrutiny. Consider detailing this softgoal, if it is under-decomposed. If the actor has no outlier softgoals, their detail is probably well balanced.
Metric	**PSWC** – **P**ercentage of **S**oftgoals **W**ith **C**ontributions
Informal definition	Percentage of softgoals with contribution links

Formal definition	`context ISTAR::PSWC()`
	`pre: self.NSIAB() > 0`
	
	`context ISTAR`
	`def:PSWC():Double = self.NSWD() / self.NSIAB()`

Requires	**NSWD** – **N**umber of **S**oftgoals **W**ith **D**ecompositions (CS) **NSIAB** – **N**umber of **S**oftgoals **I**nside **A**ctors' **B**oundaries (CS)

Concerning questions Q13 and Q14 (see tables 14 and 15, respectively), the value of PAWOUEI (percentage of actors without unconnected elements inside its boundary) and PAWDOA (percentage of actors with dependency or association links) are measures of how complete the model is and how close we are to finish the modelling process. The higher the value of these metrics, the higher the level of completeness of the model as a whole.

3.3. Example

Figure 2 shows a fragment of the Media Shop (MS) model, 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 lan-

Table 13: **Q12** – How detailed is the SR model with respect to its actors?

Symptom	The SR model has an unusually low percentage of actors with elements inside its boundary.
Potential problem	The actors specification may be over-simplistic.
Suggested action	Those actors may be good candidates for further scrutiny. Consider adding and/or detailing elements inside their boundaries. If the system has no outliers, the SR model is probably defined with the typical amount of details, with respect to the elements inside actors boundaries.
Metric	**PAWEI** – ***P**ercentage of **A**ctors **W**ith **E**lements **I**nside*
Informal definition	Percentage of actors with elements inside its boundary

Formal definition	`context ISTAR::PAWEI()`
	`pre: self.NAct() > 0`
	
	`context ISTAR`
	`def:PAWEI():Double = self.NAWEI() / self.NAct()`

Requires	**NAWEI** – **N**umber of **A**ctors **W**ith **E**lements **I**nside (CS) **NAct** – **N**umber of **A**ctors

Table 14: **Q13** – How close are we to end the assignment of responsibilities to an actor?

Symptom	The percentage of actors with unconnected elements inside their boundaries represents the percentage of actors with an incomplete specification.
Potential problem	The system specification will not be complete, which can lead to problems in its understandability. This may hamper the developers ability to faithfully implement the system according to the intention of the requirements engineer, because this intention is not documented with enough detail in the requirements model.
Suggested action	Consider completing the specification.
Metric	**PAWOUEI** – ***P**ercentage of **A**ctors **W**ith**O**ut **U**nconnected **E**lements **I**nside*
Informal definition	Percentage of actors without unconnected elements inside its boundary
Formal definition	`context ISTAR` `def:PAWOUEI():Double = 1 - self.PAWUEI()`
Requires	**PAWUEI** – **P**ercentage of **A**ctors **W**ith **U**nconnected **E**lements **I**nside (CS)

Table 15: **Q14** – How close are we to end the assignment of links to the actors?

Symptom	The percentage of actors which are not connected to other elements in the system.
Potential problem	The system specification will not be complete, which can lead to problems in its understandability. In particular, the role of the actor in the system may become unclear. This may hamper the developers ability to faithfully implement the system according to the intention of the requirements engineer, because this intention is not documented with enough detail in the requirements model.
Suggested action	Consider completing the specification by creating the necessary associations between the actor and other model elements.
Metric	**PAWDOA** – **P**ercentage of **A**ctors **W**ith **D**ependencies **O**r **A**ssociations
Informal definition	Percentage of actors with dependency or association links

Formal definition	`context ISTAR::PAWDOA()`
	`pre: NAct() > 0`
	
	`context ISTAR`
	`def:PAWDOA():Double = self.NAWDOA() / self.NAct()`

Requires	**NAWDOA** – **N**umber of **A**ctors **W**ith **D**ependencies **O**r **A**ssociations (CS) **NAct** – **N**umber of **A**ctors

guage 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 options. They can also be valuable to detect if a finished model could be more detailed, and therefore have a higher lever of completeness.

4. Evaluation

4.1. Analysed models

To evaluate the presented metrics, we used ten well-known *i** models, namely Media Shop (MS) [19], Newspaper Office (NO) [20], Health Care (HC) [3], Health Protection Agency (HPA) [21, 22], National Air Traffic Services (NATS) [22], My Courses (MC) [23], Mobile Media (MM) [24], By The Way (BTW) [25], Meeting Scheduler (MSr) [3] and Patient Wellness Tracking (PWT) [26]. We copied each of those models

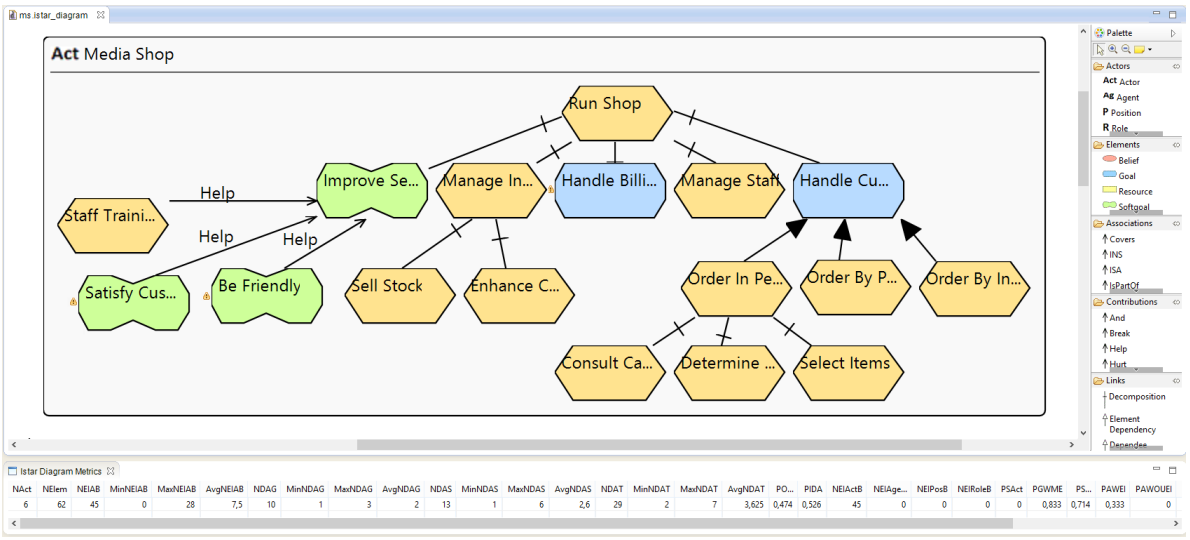


Figure 2: Application of the tool and the metrics to the Media Shop model

to our tool, and then collected the corresponding metrics. MS, NO, HC and MSr have been extensively used in the literature, while HPA and NATS are real-world systems, also documented in the literature. MC and BTW were modelled during SCORE – Student Contest on Software Engineering [27]. 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. We follow a analytical approach (as defined in [28, 29]) in this evaluation, i.e., we perform a static analysis on the structure of the selected models for static qualities (complexity and completeness).

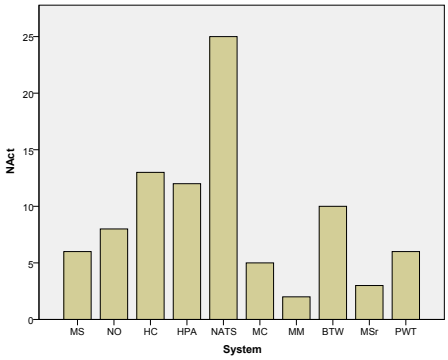
4.2. Results and discussion

In this section we present the main findings from our statistics analysis of the collected metrics. The results discussed in this section suggest a usage profile that reflects the way the different model elements are typically used in social goal models. The statistics data files and scripts for performing the statistics analysis outlined here can be found in the paper's companion site.

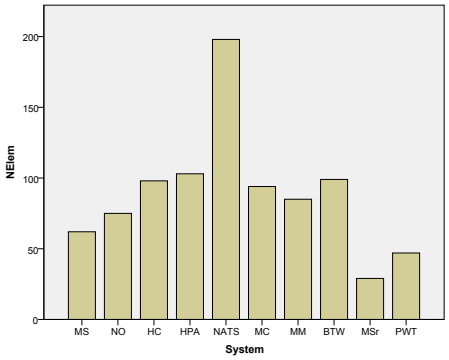
Regarding complexity, and concerning model size in particular (figure 3a and figure 3b), the NATS (National Air Traffic Services) system has, approximately, twice the size of the second and third largest systems (HC, Health Caree and HPA, Health Protection Agency). The NATS system also has the highest number of elements. The MM (Mobile Media) system presents the lower number of actors, but a number of elements similar to that of HC (Health Care), HPA (Health Protection Agency), MC (My Courses) and BTW (By The Way) systems. In figure 3c we present a boxplot chart with the distribution of the number of elements by the actors on their corresponding systems, where we can identify the outliers (denoted with O) and extremes (denoted with $*$). Even though the MM system has a significant difference between its number of elements and actors, the system has no extreme or outlier values, which suggests that the assignment of responsibilities to the actors is well balanced. Actors *e-news* (from the Newspaper Office system), *Civil ATCO* (Civilian Air Traffic Controller, from the National Air Traffic Services system) and *MyCourses* (from the My Courses system) are extremes, and the actor *Travelers* (from the By The Way system) is an outlier, which suggests that they might have too many responsibilities in the system. These actors could be interesting candidates for a decomposition into sub-actors.

This overview on complexity is just a first impression. We need to analyse more detailed features to get a clearer picture of the complexity level 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) and for each of the percentage metrics POD (percentage of outgoing dependencies) and PID (percentage of incoming dependencies), we present a boxplot chart with their distributions on the actors of their corresponding systems, in figures 3d–3h.

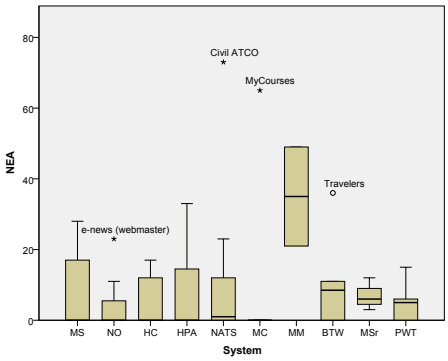
A closer inspection on the boxplot graphs for the counting metrics (figures 3d–3f) shows that there are two actors which present outlier, or even extreme values, in NDS and NDT. These should be our most likely



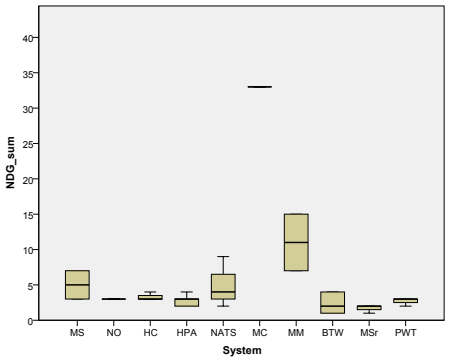
(a) NAct



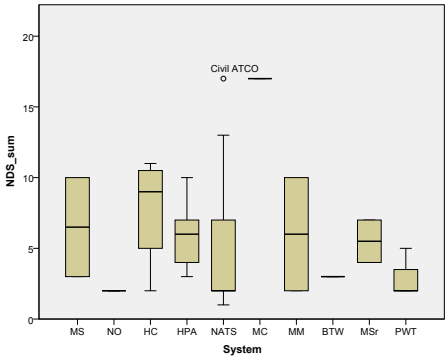
(b) NElem



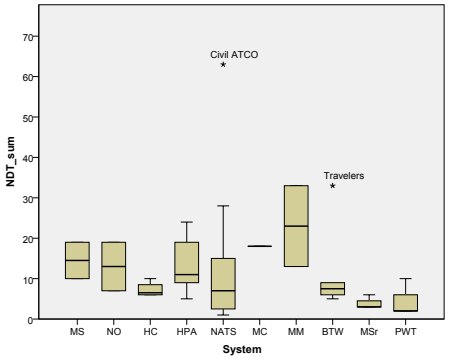
(c) NEA



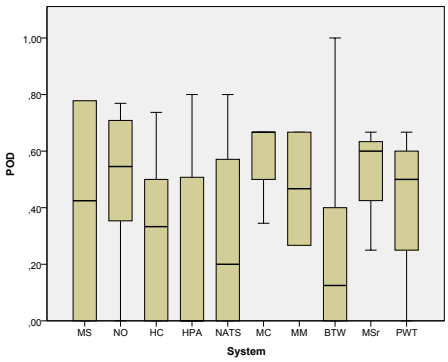
(d) NDG



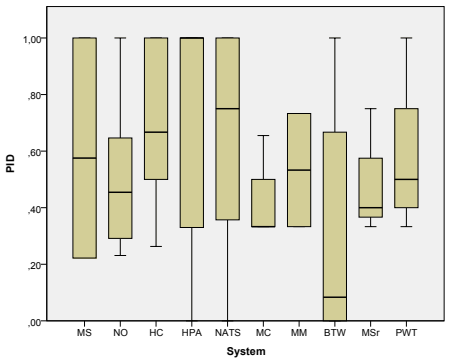
(e) NDS



(f) NDT



(g) POD



(h) PID

Figure 3: Complexity metrics values for the models

candidates for further scrutiny. For example, the actor *Civil ATCO*, from the NATS 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 solution 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 solution are similar to god classes [30] 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 a complex model. This is similar to the functional decomposition anti-pattern [30], 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 understandability of the requirements model.

A closer inspection on the boxplot graphs for the percentage metrics (figures 3g–3h) shows that the systems have no extreme or outlier values. However, the variation of the outgoing and incoming dependencies is significant between the actors of a same system. Note that in some systems, there are actors with only outgoing or only incoming dependencies. An actor with a high percentage of incoming dependencies is crucial to the system, in the sense that an expressive number of other actors are dependent on that one to achieve their goals.

In the analysed models, there are no actors with specific types, therefore, it was not possible to verify the variation in the average complexity of different types of actors. Further discussion can be found later in this section, when discussing completeness.

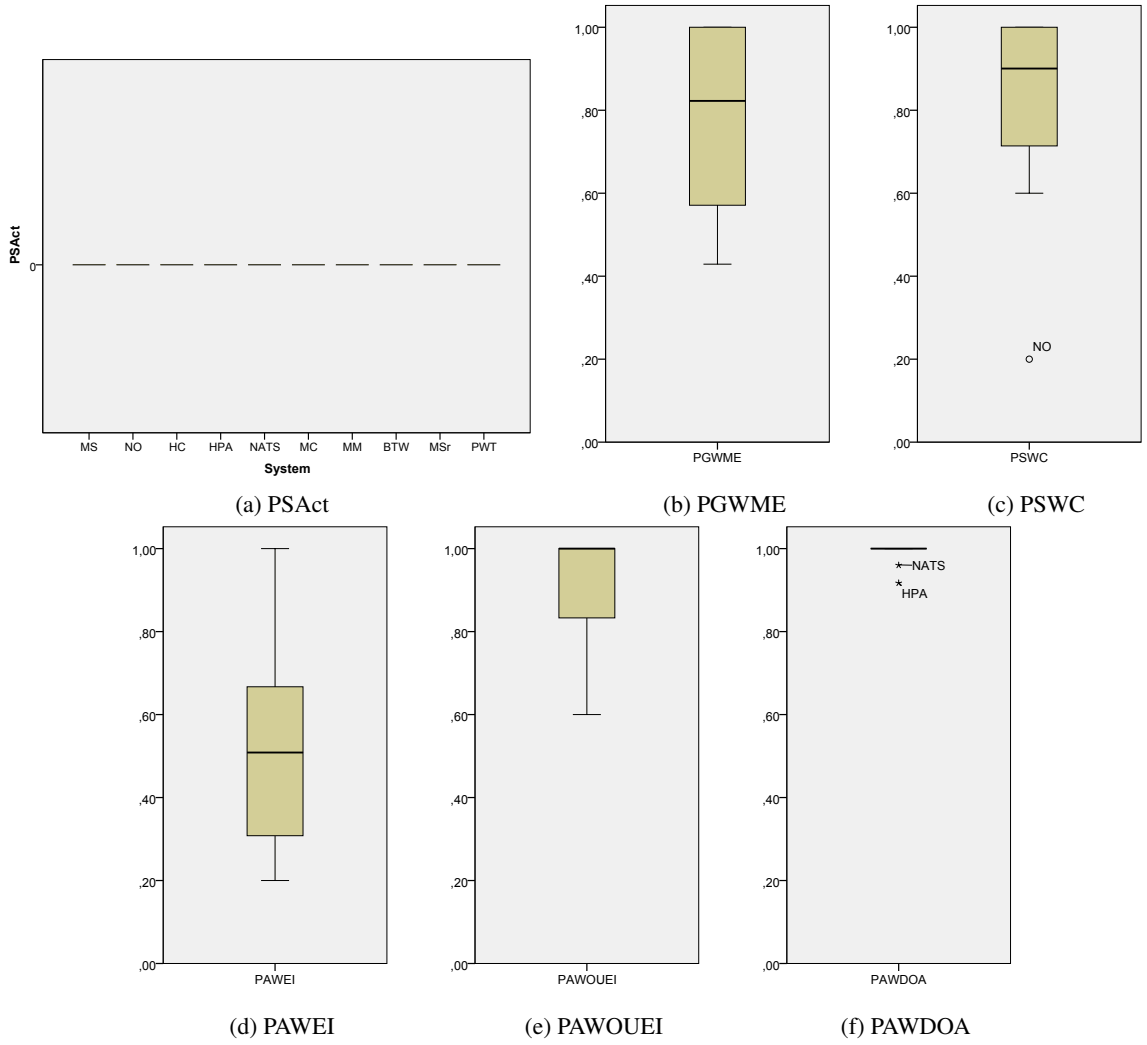
Regarding completeness, and the detail level of actors' specification in particular (figure 4a), none of the systems have an actor with a specific type. The absence of specific actors may cause the model to lose some useful information, since the level of precision and detail is lower. It may be the case where the requirements engineer has found that the usage of specific actors would not add value nor useful information to the model. Nonetheless, this absence challenges the usefulness of the actors' specification mechanism.

For each of the percentage metrics PGWME (percentage of goals with means-end links), PSWC (percentage of softgoals with contribution links), PAWEI (percentage of actors with elements inside its boundary), PAWOUEI (percentage of actors without unconnected elements inside its boundary) and PAWDOA (percentage of actors with dependency or association links), we present a boxplot chart with their distribution among the systems, in figures 4b–4f.

A closer inspection on the boxplot graphs for PGWME, PAWEI and PAWOUEI (figures 4b, 4d and 4e, respectively) shows that none of the systems have extreme or outlier values. Despite the nonexistence of these values, it is advisable that the systems are analysed in more detail and follow the guidelines presented next, since the percentage values does not reach 100% in all the systems. Regarding the percentage of goals with means-end links (figure 4b), each one of the high level functional goals needs to be refined through means-end links. Concerning the percentage of actors with elements inside, and regarding the SR model, actors without elements inside its boundary are not being sufficiently detailed, and its presence in the model may not be required, since they can not provide any relevant information. With respect to the percentage of actors without unconnected elements inside, the elements inside an actor's boundary should possess at least a link of any kind, that is, decomposition, means-end, contribution or dependency, since the presence of unconnected elements inside an actor's boundary is an indicator of incompleteness. As stated before, systems where any of these guidelines are not met, that is, whose percentage value is not 100%, should be analysed.

There are three systems which present outlier or extreme values, in PSWC and PAWDOA. These should be our most likely candidates for further scrutiny.

The percentage of softgoals with contribution links (figure 4c), presents an outlier value for the NO (Newspaper Office) system. One of the objectives of creating *i** models is to show how softgoals can be achieved through operationalization, or though more concrete actions and design decisions included in the model.



(a) PSAct (b) PGWME (c) PSWC

(d) PAWEI (e) PAWOUEI (f) PAWDOA

Figure 4: Completeness metrics values for the models

Therefore, its advisable that the requirements engineer analyses how the softgoals from the NO system could be operationalised, and add respective *contribution link*.

Concerning the percentage of actors with decomposition or association links (figure 4f), there are two extremes values, for the NATS (National Air Traffic Services) and HPA (Health Protection Agency) systems. There are two possible ways to connect actors: through dependency links and/or through association links. The assignment of links to actors can only be considered finished if the actor has at least one dependency link or one association link. Actors without any of these links are considered incomplete, since it is not clear how they are connected to the remaining actors in the system. Therefore, it is advisable that the requirements engineer analyses the NATS and HPA systems, in order to identify unconnected actors and add the respective links.

5. Related work

Horkoff and Yu [31] 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 or completeness.

Hilts and Yu [32] 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 and completeness of those existing models.

Ramos et al. [33] 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. [34] claim that GORE and MDD can be integrated to fulfil 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 [33], 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 [35] propose a framework for defining metrics in *i** models, to analyse 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 (iMDF) to better guide the analyst throughout the metrics definition process, over *i** models [36]. The method is applied to evaluate business process performance. In another follow up work, Colemer and Franch propose building *i** metrics for agile methodologies, using the iMDF method [37]. The aim of this approach is to help individuals and organisations improve requirements management activities by using metrics for agile methodologies on top of *i** models. Their approach is more focused on the process, and more generic, while we focus on modularity assessment of *i** models.

Zowghi and Gervasi [38] provided a theoretical foundation for the perspective of correctness (and its relationships regarding completeness and consistency) at requirements evolution context. The aim is to introduce more rigour into the process of requirements evolution, where the authors describe which kind of proofs must be carried out at each step during the evolution of the requirements to make sure that the final system specification satisfies the customer business goals. This work differs from ours as their contribution is on formally evaluating correctness when requirements evolve, while ours is on evaluating complexity and completeness of goal models through a set of metrics.

6. Conclusions

In this paper, we proposed a metrics suite for evaluating the complexity and completeness of *i** goal models. The proposal has been 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). They also allow evaluating the completeness of the model as a whole, and the completeness of an actor and its model elements (i.e., goals and softgoals). 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 realisation 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. Completeness analysis is useful to help requirements engineers to evaluate how close they are to completing their models. The realisation that a model is incomplete can trigger a set of model improvements, such as decomposing goals and tasks or adding elements to an empty actor boundary, increasing its detail level and easing its implementation, in the case of a software system actor. These metrics were validated by applying them to well-known industrial and academic 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 mod-

elling 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.

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

References

[1] A. van Lamsweerde, Goal-oriented requirements engineering: A guided tour, in: 5th IEEE International Symposium on Requirements Engineering, IEEE Computer Society, 2001, pp. 249–262.

[2] A. van Lamsweerde, Requirements Engineering, 1st Edition, John Wiley & Sons, Inc., 2009.

[3] E. Yu, Modelling strategic relationships for process reengineering, Ph.D. thesis, Canada (1995).

[4] ITU-T: Recommendation Z.151 (10/12), User requirements notation (URN)–language definition (2012).

[5] P. Espada, M. Goulão, J. Araújo, A framework to evaluate complexity and completeness of kaos goal model, in: 25th International Conference on Advanced Information Systems Engineering, CAiSE '13, Springer-Verlag, 2013, pp. 562–577.

[6] F. P. Brooks, The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley Publishing Company, Reading, MA, USA, 1995.

[7] ISO/IEC JTC1, OMG, Information technology – object management group object constraint language (OCL) (2012).

[8] C. Gralha, M. Goulão, J. Araújo, Identifying modularity improvement opportunities in goal-oriented requirements models, in: Advanced Information Systems Engineering, Springer International Publishing, 2014, pp. 91–104.

[9] C. Almeida, M. Goulão, J. Araújo, A systematic comparison of i* modelling tools based on syntactic and well-formedness rules, in: J. Castro, J. Horkoff, N. Maiden, E. Yu (Eds.), 6th International i* Workshop (iStar 2013), Vol. 978 of CEUR Workshop Proceedings, 2013, pp. 43–48.

[10] i* wiki, Comparing the i* tools (Last access: February 2015).
URL `http://istar.rwth-aachen.de/tiki-index.php?page=Comparing+the+i%2A+Tools`

[11] D. Kolovos, L. Rose, A. García-Domínguez, R. Paige, The Epsilon Book, Eclipse Foundation, 2013.

[12] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, Addison-Wesley Professional, 2009.

[13] Eclipse, Eclipse graphical modeling framework (gmf) tooling (Last access: February 2015).
URL `http://eclipse.org/gmf-tooling/`

[14] Eclipse, Ecore tools (Last access: February 2015).
URL `http://wiki.eclipse.org/index.php/Ecore_Tools`

[15] V. R. Basili, G. Caldiera, H. D. Rombach, The Goal Question Metric Paradigm, Encyclopedia of Software Engineering, 1st Edition, Vol. 2, John Wiley & Sons, Inc., 1994.

[16] K. El Emam, S. Benlarbi, N. Goel, S. N. Rai, The confounding effect of class size on the validity of object-oriented metrics, Software Engineering, IEEE Transactions on 27 (7) (2001) 630–650.

[17] R. Subramanyam, M. S. Krishnan, Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects, Software Engineering, IEEE Transactions on 29 (4) (2003) 297–310.

[18] i* wiki, Guideline (intermediate,layout) use the specialized actors notation to the degree that you can gain advantage in instantiating the actual stakeholders. (Last access: February 2015).
URL `http://istar.rwth-aachen.de/tiki-index.php?page=Guideline+%28Intermediate%2CLayout%29+Use+the+specialized+actors+notation+to+the+degree+that+you+can+gain+advantage+in+instantiating+the+actual+stakeholders.&structure=i%2A+Guide`

[19] J. Castro, M. Kolp, J. Mylopoulos, A requirements-driven development methodology, in: 13th International Conference on Advanced Information Systems Engineering, CAiSE '01, Springer-Verlag, 2001, pp. 108–123.

[20] C. Silva, J. Castro, P. Tedesco, I. Silva, Describing agent-oriented design patterns in tropos, in: Proceedings of the 19th Brazilian Symposium in Software Engineering, Uberlandia, Minas Gerais, Brazil, 2005, pp. 27–78.

[21] J. Engmann, Evaluating the impact of evolving requirements on wider system goals: Using i* methodology integrated with satisfaction arguments to evaluate the impact of changing requirements in hiv/aids monitoring systems in the uk, Master's thesis, England (2009).

[22] J. Lockerbie, N. A. M. Maiden, J. Engmann, D. Randall, S. Jones, D. Bush, Exploring the impact of software requirements on system-wide goals: a method using satisfaction arguments and i* goal modelling, Requirements Engineering 17 (2012) 227–254.

[23] C. Lima, J. Paes, A. Rodovalho, D. Dermeval, A. Buarque, MyCourses – A Course Scheduling System, Centro de Informática, Universidade Federal de Pernambuco, Brasil (2011).

[24] C. C. Borba, Uma abordagem orientada a objetivos para as fases de requisitos de linhas de produtos de software, Master's thesis, Brasil (2009).

[25] C. C. Borba, J. Henrique, L. Xavier, BTW – If You Go, My Advice to You, Centro de Informática, Universidade Federal de Pernambuco, Brasil (2009).

[26] Y. An, P. W. Dalrymple, M. Rogers, P. Gerrity, J. Horkoff, E. Yu, Collaborative social modeling for designing a patient wellness tracking system in a nurse-managed health care center, in: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09, Association for Computing Machinery, 2009, pp. 2:1–2:14.

[27] SCORE Contest, Score – student contest on software engineering (Last access: February 2015).
URL `http://score-contest.org/`

[28] S. T. Hevner, Alan R.and March, J. Park, S. Ram, Design science in information systems research, MIS quarterly 28 (1) (2004) 75–105.

[29] R. Wieringa, Design science methodology: principles and practice, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, ACM, 2010, pp. 493–494.

[30] W. J. Brown, R. C. Malveau, H. W. McCormick, T. J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, 1998.

[31] J. Horkoff, E. Yu, Comparison and evaluation of goal-oriented satisfaction analysis techniques, Requirements Engineering

18 (3) (2012) 199–222.

[32] A. Hilts, E. Yu, Design and evaluation of the goal-oriented design knowledge library framework, in: Proceedings of the 2012 iConference, iConference '12, Association for Computing Machinery, 2012, pp. 384–391.

[33] R. Ramos, J. Castro, J. Araújo, A. Moreira, F. Alencar, Airdoc – an approach to improve requirements documents, in: 22th Brazilian Symposium on Software Engineering, SBES, 2008.

[34] A. M. L. de Vasconcelos, J. L. de la Vara, J. Sanchez, O. Pastor, Towards cmmi-compliant business process-driven requirements engineering, in: 8th International Conference on the Quality of Information and Communications Technology, QUATIC '12, IEEE Computer Society, 2012, pp. 193–198.

[35] X. Franch, G. Grau, Towards a catalogue of patterns for defining metrics over i* models, in: 20th International Conference on Advanced Information Systems Engineering, CAiSE '08, Springer-Verlag, 2008, pp. 197–212.

[36] X. Franch, A method for the definition of metrics over i* models, in: 21st International Conference on Advanced Information Systems Engineering, CAiSE '09, Springer-Verlag, 2009, pp. 201–215.

[37] D. Colomer, X. Franch, Stargro: Building i* metrics for agile methodologies, in: F. Dalpiaz, J. Horkoff (Eds.), 7th International i* Workshop (iStar 2014), Vol. 1157 of CEUR Workshop Proceedings, 2014.

[38] D. Zowghi, V. Gervasi, On the interplay between consistency, completeness, and correctness in requirements evolution, Information and Software Technology 45 (14) (2003) 993–1009.