On the Quantitative Analysis of Agent-Oriented Models

Xavier Franch

Universitat Politècnica de Catalunya (UPC)
C/Jordi Girona 1-3, UPC-Campus Nord (Omega), Barcelona (Spain)
franch@lsi.upc.edu
http://www.lsi.upc.edu/~franch/

Abstract. Goal- and agent-oriented models are used in organization and information system modelling as a formalism aimed at providing intentional descriptions of processes as a network of relationships among actors. As such, they capture and represent goals, dependencies, intentions, beliefs, alternatives, etc., that appear in several contexts: business process reengineering, information system development, etc. In this paper, we are interested in the definition of a quantitative framework for the analysis of the properties that these models exhibit. Indicators and metrics for these properties are defined in terms of the model elements (e.g., actors, dependencies, scenario paths, etc.). Our approach is basically quantitative in nature, which allows defining indicators and metrics that can be reused in many contexts. However, a qualitative dimension can be introduced if trustable expert knowledge is available; the extent up to which quantitative and qualitative aspects are intertwined can be determined in every single case. We apply our proposal to the $i^*$ notation and we take as main case study a highly-intentional property, predictability of model elements.

1. Introduction

Goal- and agent-oriented analysis methods and languages such as KAOS, $i^*$, GRL or TROPOS [DLF93, Yu95, CKM02] are widespread in the information systems community for the refinement and decomposition of the customer needs into concrete goals, during the early phase of the requirements specification [Yu97, Lam01]. These models represent organizations and its processes as a network of actors and dependencies, which may be decomposed into simpler elements.

Once built, goal- and agent-oriented models can be used for different purposes. Two of the most important ones are: analysis of the properties they exhibit, and comparison of alternatives. In the first case, we check that some properties hold in the model, we look for some actors or dependencies that have some property (either positive or pernicious), etc. In the second case, we compare different models that represent different ways of implementing organizational processes or information systems, with respect to properties that have been considered as crucial. In both cases, evaluation of models is the cornerstone of these analyses, and therefore some suitable metrics to rely upon are needed.

The use of metrics with this purpose is common in other type of models. For instance, there are some suites of metrics in the field of object-oriented modeling
which refer to structural properties like cohesion and coupling. Properties referring to the system itself, such as security, efficiency or cost, which mainly fall into the category of non-functional or organizational requirements, appear when considering models of the system architecture [BCK03]. These metrics are usually defined in terms of the components, nodes, connections, pipes, etc., that compose the final configuration of the system.

In the case of goal- and agent-oriented modeling, most existing approaches analyse models in a qualitative way, especially in conjunction with non-functional requirements using the NFR approach [CNYM00] by targeting to specific properties such as availability, security and adaptability. These target properties are decomposed into simpler criteria that may be used to evaluate different candidate models for the system-to-be [KCM03]. This evaluation is basically qualitative, which means that the extent up to which a criterion is fulfilled by a candidate model is determined by expert judgement.

Although qualitative analysis is a powerful mechanism that may be satisfactory in many cases, it may introduce a certain degree of uncertainness because it relies completely on the claims that experts make. The dichotomy among qualitative and quantitative analysis (see [MH94, p. 40] for an abridged discussion) is not new and by no means exclusive of organization or information system modelling, or even the computer science discipline. Some researchers advocate that both types of analysis are exclusive [Sch89], but others believe that they are compatible [Pat90] and even complementary [JO04].

In this paper we are interested in the quantitative analysis of agent-oriented models. To be able to express our approach in detail, we consider agent-oriented models expressed in the \(i^*\) language, although the underlying concepts could be adapted to other approaches. More precisely, we want to take profit of the networked structure of \(i^*\) models to define structural indicators that are quantitative in nature, counting actors, dependencies, and other elements; indicators can be used to define metrics that measure model properties. Our definitions make it possible to include some expert judgement if it is considered necessary to obtain more accurate results; in fact, we show that indicators are highly customizable depending on both the knowledge we have about the problem (expert judgement and current state of refinement of the model) and the effort we want to invest in this process. Due to its structural nature, our framework is expressed in terms of the OCL [OCL03]; operators such as allInstances and select suit well for working with model elements.

The paper is structured as follows. In section 2, we define the \(i^*\) framework using UML. In section 3, we introduce our framework for measuring \(i^*\) model properties. We analyse two particular properties, predictability and segregation of duties, in sections 4 and 5, using the concepts introduced in section 3. Finally, we provide some conclusions and future work in section 6.

2. A UML Definition of \(i^*\)

We introduce next the \(i^*\) framework using the UML for defining rigorously its concepts. This section is convenient because, as reported in [Aya+05], there are several variations in the literature for the \(i^*\) notation and therefore it is necessary to make explicit which constructs do we use in this paper and which properties do we assume.
Our \( i^* \) framework is based on the seminal Yu’s proposal [Yu95] with some minor simplifications. It proposes two types of models, each corresponding to a different abstraction level (see fig. 1): a Strategic Dependency (SD) model represents the intentional level and the Strategic Rationale (SR) model represents the rational level.

A SD model consists of a set of nodes that represent actors, and a set of dependencies that represent the relationships among them, expressing that an actor (dependee) depends on others (dependees) in order to obtain some objective (dependum). Altogether form a network of knowledge that allows understanding “why” the system behaves in a particular way [Yu94]. The dependum is an intentional element that can be a resource, task, goal or softgoal (see [Yu95] for a detailed description).

A SR model allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model with reasoning capabilities. Once SR models are built, the dependencies of the SD model may be linked to the appropriate intentional elements inside the actor boundary. According to their intentional meaning, some restrictions apply: goal and task dependencies can be assigned just to goals and tasks in the dependee side; and resource dependencies just to task dependencies.

The elements inside the SR model are decomposed accordingly to two types of links:

- **Means-end links** establish that one or more intentional elements are the means that contribute to the achievement of an end. The “end” can be a goal, task, resource, or softgoal, whereas the “means” is usually a task. There is a relation OR when there are several means, which indicate the different ways to obtain the end. The possible relationships are: Goal-Task, Resource-Task, Task-Task, Softgoal-Task, Softgoal-Softgoal and Goal-Goal. In Means-end links with a softgoal as end it is possible to specify if the contribution of the means towards the end is negative or positive; this kind of contribution may also appear in softgoal dependencies.

- **Task-decomposition links** state the decomposition of a task into different intentional elements. There is a relation AND when a task is decomposed into more than one intentional element.

![Figure 1](image-url). Example of an \( i^* \) model for an academic tutoring system.
Actors can be specialized into agents, roles and positions. A position covers roles. The agents represent particular instances of people, machines or software within the organization and they occupy positions (and as a consequence, they play the roles covered by these positions).

SR models have additional elements of reasoning such as routines. A routine represents one particular course of action to attain the actor’s goal among all the existing alternatives. The concept of routine appears in [Yu95] but no notation is provided, so we use the similar notion of scenario path as defined in [LYM02] based on the use case map concept appearing in GRL [Amy99].

In Figure 2 we show the UML conceptual model [RJB04] corresponding to our version of the i* language; OCL constraints are not included for the sake of brevity. Dependencies are not defined as a ternary association but composing two binary associations to facilitate the writing of OCL expressions that appear later. We remark some modeling elements of interest: the Model class (singleton), which give a name to the model; the Node class that provides a key to model elements; the DependableNode class, which models the intentional elements for which it is possible define dependencies (actors and SR elements); the MeansEndContribution and SoftgoalContribution classes, that differentiate means-end links and dependencies that involve softgoals.

![Figure 2. A UML conceptual model for i*](image)

As an additional point, it may be argued that, in order to formulate metrics to evaluate and eventually compare i* models, it is necessary not only to rigorously define the semantics of the i* elements that we use, but also how the models are built, since different people may build correct models very dissimilar in nature and of course too much diversity would make our quantitative framework difficult to apply. We have tackled this point in our previous work, by defining two similar, complementary methodologies for building i* models, PRiM [GFM05] and RiSD [Gra+05], depending on whether the model is created as a process reengineering exercise or from the scratch,
respectively. Both methodologies define rules, checkpoints and procedures to guide model construction, therefore we may say that using them, models can be obtained in a predictable and repeatable enough manner.

3. A Framework for Metrics over $i^*$

In this section, we explore the use of structural indicators that can be used to define structural metrics that measure the properties of an $i^*$ model, i.e. those properties that depend on the form of the model and the types of its elements. Structural metrics are valuable for both analysing a highly abstract model of a system of any kind, composed basically by roles, and for comparing different feasible realizations of this abstract model (which take the form of actor models too, but including positions and agents) with respect to the most relevant criteria established in the modelled world.

Some examples of properties that appear in the literature are: a) ability, workability and commitment [Yu95]; predictability, security, adaptability, coordinability, modularity and others [KCM03]; correctness, completeness, verifiability, modifiability and traceability [KHS02]. In these and other properties, it may be the case that all the elements (actors and dependencies) influence the indicator. However, it is also possible that just elements of some particular type affect this property. Furthermore, some individual elements may be identified as especially relevant for the property; in the most general case, all the elements may have a different weight in the indicator. All these situations need to be taken into account for having a widely applicable metrics framework.

Because of this, different, related indicators can be defined accordingly to two different criteria:

- **Returned value.** We distinguish among numerical, logical and model-element indicators. Numerical indicators return a value in the interval [0, 1]; this value measures the degree of accomplishment of some criteria. Logical indicators evaluate true or false, and are used to discern if a property is fulfilled or not. Finally, model-element indicators return a (set of) model element (typically, actors, scenario paths or dependencies) that fulfils a property (e.g., scenario path that maximizes a given criteria, or set of actors that are greater than some threshold).

- **Subject of measure.** Global indicators produce a single value of any type measuring the whole model. Local indicators compute a value for any element of a given type. Group indicators calculate a value for any combination according to the grouping criteria (e.g., pairs of actors).

Therefore, given a property such as completeness, we may measure completeness of the model, of an element (e.g., an actor) or a group of related elements (e.g., all the actors of the model), with the purpose of deciding if they are complete or not, or to what extent they are complete (e.g., measuring the percentage of undefined elements) or obtaining the elements that are not complete yet. Some of the indicators can be built on top of the others, typically (but not always): logical and model-element indicators are defined in top of numerical ones; global and group metrics are defined on top of local ones.

In the next sections we analyse two properties, predictability and segregation of duties, following the concepts introduced in this section.
4. Analysing Predictability of i* Models

Predictability is used in [KCM03] as one of the properties of interest when analysing organizational styles. Its interest comes from the fact that “actors can have a high degree of autonomy depending on the way they undertake action and communication in their domains. It can be then sometimes difficult to predict individual actor characteristics as part of determining the behaviour of an organization at large” [KCM03]. Therefore, discerning up to what extent the actors of a model are predictable may be useful information for knowing more about a model.

From the several feasible points of view to analyse predictability, we opt by an external perception, i.e. how an actor perceives predictability of other actors. To be more precise, an actor is interested to know how predictable is the behaviour of those actors it depends upon, and this yields to select dependencies as the main construct of interest for defining the metrics. In the rest of the section, we first analyse predictability of individual dependencies and then we show several indicators that may be defined upon individual predictability. OCL is used for expressing predictability measures on its different forms.

4.1 Predictability of Individual Dependencies

Yu states clearly the degree of freedom bound to dependencies [Yu95, p. 15]:
- Goal dependencies. The dependee is free to, and is expected to, make whatever decisions are necessary to achieve the goal.
- Task dependencies. The depender makes the decisions, therefore the dependee cannot take a behaviour different than expected.
- Resource dependencies. They represent the finished product of some deliberation-action process, and it is assumed that there are no open issues or decisions to be addressed.
- Softgoal dependencies. The depender makes the final decision, but does so with the benefit of the dependee’s know-how.

Therefore we may conclude that task and resource dependencies are totally predictable whilst goal and softgoal ones are not. Considering that 1 represents the highest predictability and 0 the lowest, predictability of individual dependencies is defined as:

context Dependency::predictability(): Real
post: type = Task implies result = 1.0
post: type = Resource implies result = 1.0
post: type = Goal implies result = goalPredictability()
post: type = Softgoal implies result = softgoalPredictability()

To define goal and softgoal predictability we may opt among different strategies (fig. 3):
- To assign a fixed weight to all the goal and softgoal dependencies of the model. This is very basic quantitative approach, with the assumption that the factor that rules predictability is the existence of a dependency, whilst its particular meaning or hidden intentionality is not so relevant.
- To provide weights to individual dependencies by expert judgement. This option yields to a qualitative reasoning issue appearing in the context of our quantitative
procedure, which aligns with the point of view of [JO04]. This is the option to choose when just the SD model exists, which happens in the first stages of organization analysis. For instance, our RSD method [Gra+05] builds SD models from the scratch and then perform analysis before proceeding further on. At this stage, just the most relevant elements in the model exist, which means that qualitative analysis is feasible in terms of cost. Experts may use techniques such as laddering [RG88] or AHP [Saa90] as a help during their assessment.

- To find some suitable rationale for determining predictability. This alternative makes our approach basically quantitative; in fact, it may be defined in a total quantitative manner. This option seems the most appropriate when a SR model is available, which may happen in two ways: a) from the starting SD model, obtained e.g. applying RSD, dependencies and actors are refined; b) the i* model is synthesised from observation of the current organization and then the SR model exists from the very beginning, as in our PR/M method [GFM05]. This is the case we focus in the rest of the section, because it requires more decisions to be taken.

Softgoal dependency evaluation is decomposed into two factors. First, a factor bound to the depender actor, which represents how capable it is to take predictable decisions when resolving softgoals; this factor is bound to actors’ ability and not to individual softgoal dependencies. Second, a factor bound to the dependency, which represents the available know-how with respect to the given dependum. For the OCL expression, it must be taken into account that the depender can be an actor or an SR element, and in the second case its owner is obtained; a let expression makes this easier to write:

![Figure 3. Procedure for determining the Predictability of individual dependencies.](image-url)
context Dependency::softgoalPredictability(): Real
  pre: type = Softgoal
  let ownerActor(x: DependableNode): Actor =
    if x.oclIsTypeOf(Actor) then x else x.owner in
  post: result = ownerActor(depender).dependerExpertise() * knowHow()

Depender expertise may be dealt with by two different strategies: considering expert
judgement to weight individual actors, or else to agree a given weight for all the actors.
Available know-how is defined as the number of dependees that state a contribution
value to the dependum. Then, we need a function such that: 1) when the number of
contributions is 0, the function is also 0 (worst predictability because the dependees do
not know how to contribute to the softgoal); 2) as the number of contributions grow, the
function tends to 1 (best predictability). An easy, problem-independent way to define
the function is \( 1 - \left( \frac{\text{slope}}{n+1} \right) \), being \( n \) the number of contributions for the softgoal
dependum and \( \text{slope} \) a constant (defined as an attribute of the model) that determines the
slope of the function (an easy option is to define \( \text{slope} = 1 \)). Another possibility is to
define a function as a straight line from 0 to the maximum number of dependee
contributions to a softgoal dependum that exists in the model. Figure 4 shows the shape
of both types of functions.

![Figure 4](image)

**Figure 4.** Two different possibilities of know-how functions: left, inverse function with \( \text{slope} = 1 \);
right, straight line function \((n = \text{maximum number of contributions to softgoal dependum})\).

The resulting OCL definition for the inverse function case is:

```ocl
class Dependency::knowHow(): Real
  pre: self.type = Softgoal
  let theModel: Model = Model.allInstances()->any() in
  let contributionsToSoftgoalDep(d: Dependency): Integer =
    d.dependeeLink.oclAsType(SoftgoalContribution)->
    select(contr->notEmpty())->size() in
  post: result = 1 - theModel.slope / (contributionsToSoftgoalDep(self)+1)
```

Fig. 5 presents an example of this case. It is an excerpt of a model for a distance
learning environment. The dean has as one of her goals to achieve academic quality, and
for this goal she depends on teachers and tutors for having *Good Course Dynamics*.
There are several ways in which teachers may contribute positively to this softgoal:
publishing exams’ marks timely, answering students’ messages daily and making FAQs
lists available. An important issue that influences course dynamics in distance learning
is the feedback that teachers provide to students about their exams. There are roughly
two strategies: sending personalized messages to students commenting their mistakes, or
just giving group support by making public the solution and the evaluation criteria, and
sending personalized information just on demand. The first strategy is considered to
impact positively into the dynamics of the course, but not the second. Concerning tutors,
it has not been investigated yet how can they contribute to course dynamics. As there
are 5 contributions to the softgoal dependency, the OCL definition above with Model.slope = 1 yields GoodCourseDynamics.knowHow() = 1 - (1 / 5+1) = 0.83. Since the dean is a highly strategic actor, we consider dependerExpertise() = 1.0 and therefore GoodCourseDynamics.softgoalPredictability() = 0.83.

![Diagram](image_url)

**Figure 5.** Distance learning environment model: predictability of softgoal dependencies.

Concerning goal dependencies, unpredictability depends on how many ways the dependees have to fulfil the goal. As stated in section 2, a goal dependency may have just goals and tasks as dependees. In both cases, we compute the number of different task combinations that may attain the goal, by descending from the goal or task, using means-end and tasks decompositions: the more combinations there are, the less predictable is the dependee with respect to that dependency. It is worth to remark that if the dependency involves more than one dependee, unpredictability is present from the very beginning, because this means that there are many ways to attain the goal. It is necessary also to consider the case that the dependee is not a SR element but an actor, which means that the dependency has not been assigned yet to an intentional element and thus unpredictability is the highest (i.e., equals to zero).

Similarly to the case above, we define a problem-independent function as the inverse of the number of combinations. The corresponding OCL function is outlined below, not including the function that computes the number of combinations, because it takes a significant number of lines:

```ocl
class Dependency:
  goalPredictability(): Real
pre: self.type = Goal
let nbTaskCombinations(d: Dependency) = ... in
post: nbTaskCombinations(self) = 0 implies result = 0
post: nbTaskCombinations(self) > 0 implies
  result = 1 / nbTaskCombinations(self)
```

Fig. 6 presents an example of this case focusing on how exam evaluation feedback is provided. Two goals already introduced in fig. 5 are refined. The most general goal that appears, Exam Feedback Provided, is the dependee of the student’s goal Feedback from
Exams Acquired. Since this goal has two means-end decomposition (which are implicitly OR-ed, see section 2), there are two different ways to provide feedback. Therefore, the evaluation for this dependency is $\frac{1}{2} = 0.5$. Effects of unpredictability are clear if we analyse how the elements that appear in the decomposition relate to other model elements. For instance, Personalized Feedback Provided has a negative contribution to the Personal Workload kept Low softgoal that the teacher has. This contribution is stating that deciding among Personalized or Group Feedback Provided depends on what the teacher considers a reasonable threshold for her workload, and since this is out of the student’s control, predictability gets damaged.

As a final remark, we would like to point out that the obtained indicator for dependency predictability is highly customizable (therefore reusable and repeatable); key points are: does the SR model exists or not?, do I really need expert judgement or do I keep my approach purely quantitative?, if expert judgement chosen, do I prefer to weight individual elements or do I give the same weight to all of them? The procedure depicted at fig. 3 shows clearly the needed steps, in the figure, the information required during the process is represented by underlined italics in the body of OCL expressions.

Figure 6. Distance learning environment model: predictability of goal dependencies.

4.2 Indicators for Predictability

Different indicators may be defined on top of dependency evaluation along the two dimensions presented in section 3. Of particular interest is the dimension about the subject of measure. Three feasible possibilities are:

- Analyse predictability of actors. Two different points of view are possible: how predictable an actor perceives its environment, and how predictable an actor looks to its environment. In the first case, the focus is on the dependencies in which the actor acts as depender, whilst in the second case the target is the dependee side. For instance, for the first point of view:
context Actor::perceivedPredictability(): Real
let actorDependencies(a: Actor): Set(Dependency) =
    Dependency.allInstances()->
    select(d | d.depender = a or d.depender.owner = a)
in post: actorDependencies(self)->size() = 0 implies result = 1
post: actorDependencies(self)->size() > 0 implies
    result = actorDependencies(self).predictability()->sum() / actorDependencies(self)->size()

• Concentrate on scenario paths, as representative of business processes. Scenario
paths are composed by steps that are tasks or goals. Each step is either decomposed
inside the boundaries of the actor or as depending on external actors; this two cases
rule the OCL decomposition below. In both cases, predictability depends on the
number of task combinations that exist to carry out the step:
context ScenarioPath::predictability(): Real
post: result = step.predictability()->sum() / step->size()
context TaskOrGoal::predictability(): Real
let dependsUpon(): Boolean = self.dependency[depender]->notEmpty()
in post: dependsUpon() implies
    result = dependency[depender].predictability()->sum() / dependency[depender]->size()
post: not dependsUpon() and nbTaskCombinations() = 0
    implies result = if type = task then 1 else 0
post: not dependsUpon() and nbTaskCombinations() > 0
    implies result = 1 / nbTaskCombinations(self)

being TaskOrGoal::nbTaskCombinations() a function that computes the
number of task combinations for that task or goal, defined analogously to
Dependency::nbTaskCombinations() introduced in section 4.1.

• Define predictability for the whole model as done in [KCM03], obtaining therefore a
single value. They use this property to compare different organizational patterns such
as joint venture, structure in 5, and others:
context Model::predictability(): Real
post: result = Dependency.allInstances().predictability()->sum() / Dependency.allInstances()->size()

Concerning the second dimension, these numerical indicators can be used to obtain
boolean or model elements ones, allowing e.g.: finding out if strategic actors exceed
some threshold; given two models, which one is the most predictable; ordering all the
actors in terms of predictability; checking that scenario paths are fully predictable; etc.

5. Segregation of Duties

To provide more insights to our framework, we address to a completely different kind of
property, closer to the organizational world than predictability (which is more a
modelling-related property). The interest is not only this different orientation of the
property to be measured, but also to illustrate how indicators analysis may help to tune
the organizational i* model (and therefore the business processes themselves) to fulfil
properties of interest.
One key aspect on organization analysis is how organizations comply with regulations stated by laws, best practices, etc. Nowadays, one law that has gained importance in the US is the Sarbanes-Oxley Act (2002) that has to be with the separation of concerns when implementing business process, which is called segregation of duties (SoD). SoD is the result of applying policies that ensure the separation of incompatible business duties and/or responsibilities in a business process [Bur04, Tar04]. Ensuring that business processes are compliant with the SoD principle is currently an extended practice, and therefore one could wonder if indicators can be defined over our / models to check SoD.

The exact definition of SoD is highly context-dependent. If the focus is on business transactional processes, it is widely recognised that four general categories of duties exist, which may create conflict when assigned to the same employee: authorization of the transaction, custody of assets, record keeping and reconciliation when conflicts appear. In a single organization, many situations that demand SoD may arise, some according to this classification, and others more vertical oriented, i.e. very specific of the organization domain. For this reason, defining general, reusable indicators for SoD require some configuration to be left when using them in a concrete setting.

Since business processes are the atomic unit for this concept, it seems adequate to adopt a logical indicator using scenario paths as subject of measure for SoD. It is necessary to keep in mind that having the conflicting tasks in different actors is a necessary but not sufficient condition for segregation of duties; it is also necessary to ensure that positions do not cover simultaneously two roles that have the tasks assigned, and that agents do not occupy simultaneously two positions that have the tasks assigned:

context ScenarioPath::segregationOfDuties(): Boolean
post: result = forAll(t1, t2: TaskOrGoal | t1 <> t2 and toBeSegregatedDuties(t1, t2) implies t1.responsible() -> intersection(t2.responsible())->isEmpty()

context TaskOrGoal::responsible(): Set(Actor)
post: not owner.oclIsTypeOf(Role) and not owner.oclIsTypeOf(Position) and not owner.oclIsTypeOf(Agent) implies result = owner
post: owner.oclIsTypeOf(Agent) implies result = owner
post: owner.oclIsTypeOf(Position) implies result = owner->union(owner->agent)
post: owner.oclIsTypeOf(Role) implies result = owner->union(owner->position)->union(owner->agent)

The function toBeSegregatedDuties evaluates true if the compared steps shouldn’t be assigned to the same actor. It is part of the expert judgement to decide which steps of the routine should be segregated, which may not be an easy task [HT05].

In fig. 7 we show how this indicator may be used in our distance learning example. When modelling the organization for analysing business processes, agents are rare in the model; roles and positions are prevalent. In one of the routines, there are two tasks that the expert thinks must be segregated, Evaluate Exams and Resolve Conflicts. In the model on the left, both tasks appear assigned to the same actor; thus, SoD is violated. From this observation, we realize that the organization must change, distinguishing two different positions, one for teachers on charge of evaluations and the other for teachers responsible of conflict solving (fig. 7, middle), therefore tasks become effectively segregated. However SoD must be checked again when agents are added in the model; fig. 7 at the right shows an agent assignment that makes SoD to be violated again.
6. Conclusions and Future Work

In this paper we have presented a framework for the definition of structural metrics for agent-oriented models using the $i^*$ language. The metrics are bound to properties of the system model, which usually represent correctness concerns, organizational issues or information systems requirements. The framework considers the definition of indicators organized according to two dimensions (returned value and subject of measure). The indicators are customised to use expert judgement as considered necessary, although we may say that they are basically quantitative in nature. We have shown how these indicators may be used to find out properties of the system, and even their use as a way to check and eventually change the structure and business processes of organizations.

The most relevant characteristics of our approach are:

- **Accuracy.** We have provided a UML definition of $i^*$ models that is used as a baseline upon which we have build our framework. Indicators and metrics are expressed with the OCL. Although not shown in the paper, we have methodologies to drive the construction of $i^*$ models in a consistent way.

- **Expressiveness.** The use of the OCL allows expressing metrics both in a comfortable and expressive way. Comfortability, as shown in the predictability example, comes from the object-orientation paradigm that appears in the heart of our framework.

- **Sensitivity.** Metrics can be defined more or less accurately depending on: 1) the expert judgement available; 2) the state of refinement of the model; 3) the effort allocated to model analysis. Therefore, we have a highly configurable framework that allows defining metrics in several ways (see fig. 3 as an example).

- **Easy tool support.** The form that our framework takes allows implementation of tool support to drive indicators definition, model edition, generation of organizational alternatives, and evaluation of models. We have a first prototype deployed as a plug-in of the REDEPEND modelling tool [GFM05b] which uses metrics patterns as a way to improve productivity.

- **Reusability.** As shown in the analysed properties, the indicators and metrics obtained are independent of the domain and therefore applicable to any model.

In the introduction we have mentioned the existence of qualitative approaches for analysing $i^*$ models but, to the best of our knowledge, there is not much related work...
from a quantitative point of view. The most comprehensive proposal in this area we know about exists as part of the AGORA method [KHS02] that provides techniques for estimating the quality of requirements specifications in a goal-oriented setting. In fact, AGORA puts more emphasis in the analysis of the AND/OR graph resulting from decomposition than in the kind of analysis that has been the focus of this paper. Therefore, comparison is not really possible and in fact, we could think of using AGORA and our approach jointly. Also, it is worth mentioning [SM99] which proposes the analysis of dependency coupling for detecting excessive interaction among users and systems. They use expert judgement to classify the dependencies of the system in a qualitative scale and then define a metric on the model that use to compare alternative scenario. This metric for coupling is a good example of structural metric and we can check that it is definable using our framework in a straightforward way.

We have identified several ways to proceed ahead in this line of research. For making our proposal useful, we remark the following:

- Construction of a catalogue of reusable indicators and metrics. Basically in three directions: 1) model-related properties; 2) organizational-related properties; 3) properties addressing non-functional aspects such as security, efficiency and so on.
- Identification of patterns for indicators and metrics. We have realized that most of the indicators and metrics definitions apply similar rules over and over. In [FGQ04] we have identified some patterns that capture some of these situations and we plan to enlarge the catalogue.
- Better tool-support. We plan to enlarge our current prototype and adapt it to the OCL as metrics definition language (currently we work with an ad-hoc notation). Also we aim at defining wizards that help to implement indicators and metrics definition asking required information, as illustrated in fig. 3.
- Integration of the framework with other proposals. In particular, we are especially interested in using this framework in the analysis of system architectures [BCK03].
- Validation plan for the framework. We have carried out a preliminary validation based on the properties and models defined in [KCM03]. The validation has not been easy because models were not directly comparable, therefore we have reworked them and formulated i* SR decompositions.

References


