Towards a Catalogue of Patterns for Defining Metrics over *i** Models

Xavier Franch and Gemma Grau

Universitat Politècnica de Catalunya (UPC)
UPC – Campus Nord, Omega building, 08034 Barcelona, Spain
franch@lsi.upc.edu, ggrau@lsi.upc.edu

Abstract. Metrics applied at the early stages of the Information Systems development process are useful for assessing further decisions. Agent-oriented models provide descriptions of processes as a network of relationships among actors and their analysis allows discerning whether a model fulfils some required properties, or comparing models according to some criteria. In this paper, we adopt metrics to drive this analysis and we propose the use of patterns to design these metrics, with emphasis in their definition over *i** models. Patterns are organized in the form of a catalogue structured along several dimensions, and expressed using a template. The patterns and the metrics are written using OCL expressions defined over a UML conceptual data model for *i**. As a result, we promote reusability improving the metrics definition process in terms of accuracy and efficiency of the process.

1 Introduction

Measuring is a central task in the Information Systems (IS) development process. Some measures are used to evaluate an already built IS, for instance, by establishing its size according to the number of classes or lines of code, or by checking that the resulting system accomplishes its non-functional requirements fit criteria. However, measures can also be taken at the early stages of the IS development process, where they allow predicting some of the quality factors of the system-to-be, and planning corrective actions if needed. Therefore, there are many approaches that propose metrics over specification artefacts such as Statechart Diagrams [14], Use Cases [28] or OCL expressions [27]. Other metrics focus on the business process using Workflow Diagrams [26] or propose a mixed approach using Maps for representing the business process and UML class and transition diagrams for representing the system [8]. In this paper we are interested in the definition of metrics over a particular type of specification artefacts, namely goal-oriented models [24] expressed using the *i** framework.

The *i** framework [33] is currently one of the most widespread goal-oriented modelling and reasoning frameworks in IS development. One of its strengths is that it proposes different kinds of constructs that allow representing in a single model both the strategic needs of the business process and the operational specification of the IS. Because of that, it is currently used in different disciplines such as requirements

Z. Bellahsène and M. Léonard (Eds.): CAiSE 2008, LNCS 5074, pp. 197–212, 2008.

engineering [12], [25], business process modelling and reengineering [32], organizational modelling [23], or architecture representation [11], among others. As a consequence of this intensive work, the *i** framework has gained a solid position in the community and several research results can be considered as consolidated such as ontological issues [20], [21], metamodelling [30], development methods [2], [19], or reasoning algorithms [15], [29]. But on the other hand, this continuous research issues many new challenges to overcome, being one of them the analysis of IS represented with *i** models. One way to conduct such an analysis is by using metrics. Having good suites of metrics allows not only analysing the quality of an individual model, but also comparing different alternative models with respect to some properties in order to select the most appropriate alternative. Therefore, we claim that having a comprehensive catalogue of *i** metrics would be a significant contribution to the state of the art in *i** and, therefore, for IS development.

Aligning with this belief, in our group, we are using metrics on *i** models from some time ago [10], [9], [19] and we have formulated several metrics in different disciplines (see Section 2). One of our current lines of research is constructing a comprehensive framework of metrics over *i** models. During this research, we are facing two particular problems: (1) for some concepts to be measured, the expression of the metrics is cumbersome; (2) even in metrics with quite different purpose, there are some elements in the process that appear over and over, mainly the type and combination tactic of *i** elements to conform the metrics. The purpose of this paper is to formulate the basis of a catalogue of metrics over *i** models that helps to overcome the two problems mentioned above. With this aim, we are presenting a catalogue of patterns aimed at supporting the definition of metrics over *i** models.

The remainder of the paper is organized as follows. In section 2 we present an overview of the antecedents on metrics over *i** models and in section 3 we present a metamodel for the *i** framework (basic knowledge on *i** is assumed). Our final purpose is to define a catalogue of patterns to be used when defining metrics over *i** models and, for doing this, in section 4 we propose a template for documenting the patterns. In order to guide their use, patterns can be classified into several categories by forming a structured catalogue, which we present in section 5. In section 6 we show an example of application. Finally, we end with the assessment, the conclusions and future work in section 7.

2 Antecedents

Roughly speaking, we may classify existing proposals for analysing *i** models into quantitative and qualitative depending on the dominant dimension. In the quantitative-dominant side, we mention the AGORA method [22] that provides techniques for estimating the quality of requirements specifications with emphasis in the AND/OR decomposition of goals. Sutcliffe and Minocha [31] propose the analysis of dependency coupling for detecting excessive interaction among users and systems; they combine quantitative formulae based in the form of the model with some expert judgment for classifying dependencies into a qualitative scale. Bryl *et al.* propose structural metrics for measuring the Overall Plan Cost of an agent-based system [5]. For qualitative-predominant techniques, we mention Yu's seminal contribution [33]

concerning ability, workability and commitment, and also [15], [29], which combine qualitative assessment with some rules for forward and backward propagation over AND/OR decomposition graphs in the context goal satisfaction analysis.

Concerning our own work, our first proposal appeared in [11]. In this paper, we used *i** to model component-based software architectures and defined six different metrics aimed at informing the selection of the most appropriate architecture with respect to some software requirements: diversity, vulnerability, packaging, data self-containment, uniformity and connectivity. Then, in [10] we defined a general framework. We focused on the following metrics: data privacy, data accuracy, process agility and responsibility dissolution. This framework was formalised in [9], defining a UML metamodel and using OCL as metrics definition language. We explored as metrics: predictability of models, and segregation of duties (in the extended version of [9] as technical report). Finally, in [17] and [18] we applied our approach in reengineering processes obtained by customization of ReeF, a generic Reengineering Framework [16]. In [17], we explored reengineering of software architectures over a documented case study and for this purpose we defined over the *i** framework two classical metrics, coupling and cohesion. In [18], we targeted reengineering of software processes and we focused mainly on defining the functional size of a software system in the COSMIC-FFP framework (using then cfsu, COSMIC functional size unit, as metric); we also included some results about process agility and ease of communication in the considered organizational alternatives. Other metrics are not currently available in the form of publications.

From ours and others' previous research, we observed the following facts:

- *i** is a versatile framework to represent concepts at different levels of abstraction (organizations, processes, architectures, etc.) and allows stating pertinent metrics over the models built. However, for some metrics, some additional information is required, as the type of actor, the criticality of a resource or task, etc.
- Purely qualitative or quantitative approaches do not exist, therefore we must be able to combine techniques of both kinds in the framework to make it usable.
- The process of definition of metrics in *i** may be cumbersome. An example is actors' predictability as defined in [9], which required six OCL predicates along with some auxiliary let-expressions for its definition.
- However, even in metrics with quite different purpose, there are some elements in the process that appear over and over, making thus both feasible and convenient to define a catalogue of patterns for defining *i** metrics.

3 A Metamodel for the *i** Framework

The *i** framework is both a goal- and agent-oriented framework with the aim of modelling and reasoning about organizational environments and their information systems. For doing so, it offers a formal representation of the involved actors and their behaviour. *Actors* can be specialized into *agents*, *roles* and *positions*. A position *covers* roles. The agents represent instances of actors within the organization and they *occupy* positions (consequently, they *play* the roles covered by positions).

The *i** framework proposes two types of models for modelling systems each one corresponding to a different level of abstraction: the Strategic Dependency (SD) and

the Strategic Rationale (SR). An SD model consists of a set of nodes that represent actors and a set of dependencies that represent the relationships among them. Dependencies express that an actor (*depender*) depends on some other (*dependee*) in order to obtain some objective (*dependum*). Depending on the *dependum* kind, the *depender* depends on the *dependee* to bring about a certain state in the world (goal dependency), to attain a goal in a particular way (task dependency), for the availability of a physical or informational entity (resource dependency) or to meet some non-functional requirement (softgoal dependency).

An SR model allows visualizing the intentional elements into the *boundary* of an actor in order to refine the SD model with reasoning capabilities. The dependencies of the SD model can be linked to the appropriate intentional elements (also classified as goals, softgoals, tasks and resources) inside the actor boundary. The elements inside the SR model are decomposed accordingly to three 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. In *Means-endContribution* links, with a *softgoal* as end, it is possible to specify if the contribution of the means towards the end is negative or positive. *Task-decomposition links* state the decomposition of a task into different intentional elements. We refer to [1] for details about usage of links. *Scenario paths* (also called routines in [33]) are composed of tasks and goals. For more details on the *i** framework, we refer to [33].

In Fig. 1 we present a metamodel for the *i** framework that represents the explained concepts that will be used for defining metrics. The metamodel is essentially the same as in [9] (which in its turn is similar to other existing proposals) but including a

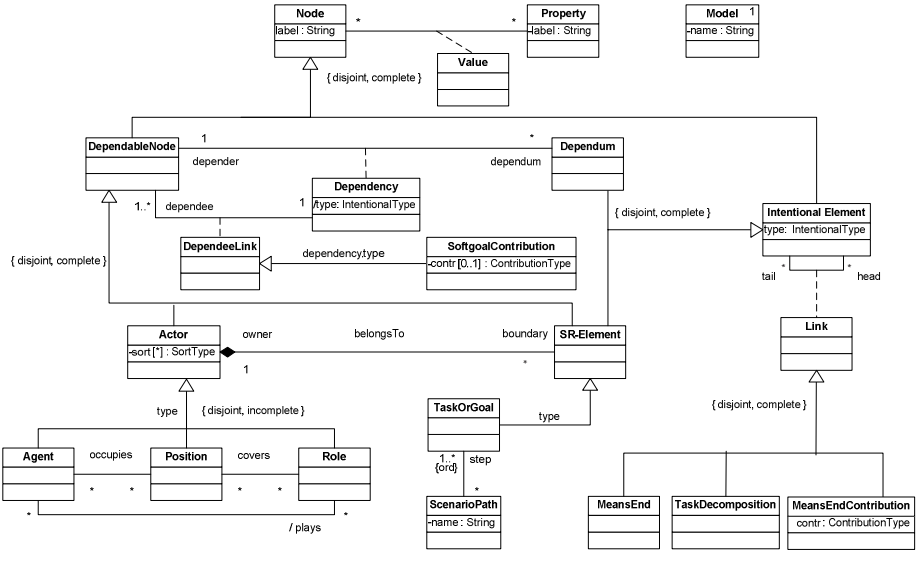

Fig. 1. A UML class diagram for *i** (some specializations are omitted due to the lack of space)

Table 1. Example of pattern: the Sum pattern

Name	**Sum**
Context	A metric is defined over two types of different model elements such that elements of one type (aggregated) contain elements of the other (aggregee)
Problem	There is a need of computing the aggregated metric in terms of aggregee's
Solution	Define aggregate's metric, Aggregated::metric, as the sum of aggregee's
Involved Classes and Types	Aggregated: <<Node>> -- aggregated's class in the *i** metamodel Aggregee: <<Node>> -- aggregee's class in the *i** metamodel Type: <<DataType>> -- the type of the metric
Assumptions	– The metric ranges onto a numerical data type – The Aggregated class is an aggregation (either direct or transitive) of Aggregee – There is a definition of the metric over the Aggregee, Aggregee::metrics
Required Knowledge	– The relationship of aggregation from Aggregated to Aggregee, Aggregated::aggregees(): Set(Agregee)
Form	**context** Aggregated::metric(): Type **post**: result = self.aggregees().metric()->sum()
Related Patterns	– Numerical patterns (e.g., Normalization) to manipulate the result – Navigational patterns (e.g., All Elements of a Kind) to define aggregees – Discrimination patterns (e.g., Discrimination By Type) to filter the aggregees
Example of Use	In the context of summing the size of the resources managed by an actor: **context** Actor::size(): Integer **post** result = self.allResources().size()->sum()

singleton class for models, types of actors, and the possibility to attach properties that may influence metrics to model elements, improving thus expressiveness. OCL integrity constraints are not included for the sake of brevity.

4 The Pattern Template

As mentioned in section 1, when analysing *i**-based metrics, we observed some patterns that appeared over and over. Their identification and classification would help in metrics definition and reuse. In this section we define the information enclosed in patterns definition, which yields to a proposal of template.

The structure of the pattern is as follows:

- **Name**, **context**, **problem**, **solution**, **related patterns**, and **example of use.** These are usual components of every pattern-based proposal such as [13] and have proven their usefulness for documentation purposes.
- **Involved classes** and **types.** Elements from the *i** metamodel that appear in the definition of the pattern. Also, the data types required appear here.
- **Assumptions.** The concept of assumption as proposed in [4] is included in order to embody the intuitive knowledge about the pattern. Assumptions provide the basis over which the metrics are defined.
- **Required knowledge.** Domain-related information that has to be provided in order to effectively use the pattern.
- **Form.** An OCL expression that defines the pattern in terms of the involved classes and types, the required knowledge and under the stated assumptions.

Table 1 shows an example of the documentation template for the Sum pattern. This pattern computes the value of a metric applied on a model element (*aggregated*) in

terms of the same metric applied on some other model element (*aggregee*) which is related with the former by an aggregation relationship (*aggregees*). This is one of the most used patterns when a metric is decomposed top-down, e.g. a model metric as the sum of the metrics applied to its actors, or an actor metric as the sum of the metrics applied to its intentional elements. Often, the result is modified with some Numerical pattern (e.g., the result may be normalized into a given interval). Also, the aggregee's metric is sometimes applied just to those aggregees that satisfy some property (e.g., software actors, or resource intentional elements) using some Discrimination pattern. Last, some Navigational templates may be used to generate the *aggregates* function, as it would happen for the allResources that appears in Table 1. These types of related patterns are presented in the next section.

A concept that is fundamental when using the patterns is that of pattern instantiation. When a pattern is fully instantiated, it becomes an OCL expression completely determined, ready to be evaluated. A complete and correct instantiation requires specifying which actual model elements, types and knowledge play the parts identified in the pattern, which of course must fulfil the stated assumptions. A complete and correct instantiation of the Sum pattern presented in Table 1, to obtain the metric defined in the Exemple of Use part of the template, would be declared as:

size ::= Sum[Aggregated ::= Actor, Aggregee ::= SR-element, Type ::= Integer,
aggregees() ::= allResources()]

5 The Catalogue of Patterns

In order to facilitate the definition and reuse of the proposed patterns, we adhere to the catalogue definition stated in [13], which includes classification criteria as part of the pattern. From our experience, we have identified a four-dimension classification of the patterns, organized into categories and subcategories.

Table 2 presents these main categories and subcategories as well as s representative sample of patterns (we have near one hundred of such patterns and therefore it is not feasible to present all of them here). In the process of conforming metrics, several of these patterns can be instantiated either sequentially or nested. The definitions are provided below (together with some examples; we skip the Examples of Use clause since all these patterns are used in the example of section 6):

- **Metrics declaration.** The first decision to take when conforming metrics is to decide their concrete form. We have two different criteria to identify patterns:
  - The subject of measure. The metrics may apply to the whole model, an individual model element (e.g., an actor; see Table 3) or in the middle, a set of model elements (e.g., metrics to analyse pairs of actors). Depending on this granularity, it is possible to determine the context of the OCL expression used for the metric.
  - The objective of the metrics. Its effect is to determine the OCL expression return value's type: enumeration or string; numerical; Boolean; or aggregation of model elements. It may be a classification instrument (i.e., a nominal metric), a measuring instrument (from an ordinal, absolute or ratio scale), a condition-checker (checking if a given domain property is attained) or a

locator (searching for a model element, or aggregate of model elements, that satisfy a condition). A measuring instrument may be used as the basis to obtain instruments or the other type. For instance, a Boolean metric may be defined as a numerical metric compared to a certain threshold value; or a sequence of model elements may be ordered with respect to the numerical metric value.

Table 2. Overview of the proposed catalogue of patterns

Category	Subcategories			Pattern
Metrics Declaration	Subject			Model
				Set of model elements
				Individual Element
	Result			Classification Instrument
				Measuring Instrument
				Condition-Checker
				Locator
Metrics Definition	Qualitative			By Criterion
				Individual
				Global Information
	Quantitative	Structural	Aggregation	Sum
				Count
			Discrimination	By Type
				By Type and Value
			Element-Based	Actor-Based
				Dependency-Based
		Property-Based		
Metrics Transformation	Numerical	Inverse		
		Average		
		Normalization		
Metrics Auxiliary Elements	Navigational	All Elements of a Kind		
		Superclass		
		Transitive Clousure		

- **Metrics core definition.** To define the metrics (i.e., the body of the OCL expression), the discussion about qualitative and quantitative predominance issued in section 2 drives our further classification:
  - Qualitative-predominant definition. Their use appears when the metric is strongly domain-dependent, or manages concepts that do not appear explicitly in the model (e.g., actors' tacit knowledge). In these cases, a domain expert must provide the needed knowledge. We have mainly three subcategories:
    - ◊ By criterion. Each model element that satisfies some criterion (e.g., being a resource dependency, or a human actor) has a given value assigned.
    - ◊ Individual. Each model element has a value assigned (see Table 4).
    - ◊ Global information. A particular item of information which represents global knowledge that may affect many metrics. This information is considered as owned by the Model single instance (i.e., it may be considered as an attribute of the class Model).
  - Quantitative-predominant definition. The model encloses all the necessary information for computing the metrics. We distinguish two big subcategories:

◊ Structural. The metric is computed from the form of the model, using its structure: actors, dependencies, etc. There are quite a lot of patterns belonging to this category, see Table 2 for some of them. This subcategory is divided into three. Aggregative patterns generalize the idea explained in section 3 for the Sum pattern, which in fact is a particular case, as the Count pattern, which are the two most used aggregative patterns. Discrimination patterns use some criteria on model elements to compute the metric (see Table 5). Element-based patterns define a metric on a basic element of the model, typically dependencies or actors (see Table 6).

◊ Property-based. The metric uses domain properties that are embodied in the model using the *property* operation from the Node class.

- **Metrics transformation.** They modify the value by applying some transformation, usually numerical. Typical examples are the inverse function and the average and normalization within a range (see Table 7).
- **Metrics auxiliary elements.** They capture repetitive situations when defining metrics. The most significant subcategory is navigational patterns, which provide OCL expression types to navigate and obtain (an aggregate of) a model element (e.g., all the tasks that form a routine, or a dependency's depender).

Categories may have a template bound, which makes possible to implement the catalogue hierarchy by using the concept of pattern specialization: a subcategory

Table 3. Example of pattern: the Subject->Individual Element pattern

Name	**Individual element (Metrics Declaration -> Subject)**
Context	The process of conforming a metric has just started
Problem	The metric has no sense for most types of model elements except one
Solution	Define the metric with this type of element as context
Involved …	Elem: <<Node>> -- class corresponding to this type of element in the *i** metamodel
Assumptions	N/A
Required …	N/A
Form	**context** Elem::metric(): STILL TO KNOW **post**: STILL TO KNOW
Related patterns	– Result patterns (e.g., Locator) to declare the result – Metrics definition patterns to complete the definition

Table 4. Example of pattern: the Qualitative->Individual pattern

Name	**Individual (Metrics Definition -> Qualitative)**
Context	In the process of defining a metric, a particular type of model element must be assessed
Problem	– Just the SD model is available (not the SR) and this implies lack of information, or – Quantitative analysis may be unacceptably costly or not feasible
Solution	Provide individual qualitative assessment for each model element
Involved …	Type: <<DataType>> -- the type of the metric
Assumptions	N/A
Required knowledge	function to represent expert judgement on each individual model element of the type, judgement: String → Type such that domain(judgement) = Node.allInstances().label
Form	**context** Node::ExpertJudgement(): Type **post**: result = judgement(self.label)

Table 5. Example of pattern: the Discrimination by Type pattern

Name	**By Type (Metrics Definition -> Quantitative -> Structural -> Discrimination)**
Context	Some metrics that are defined over one particular type of node (e.g., actor, dependency, SR-element) have a value that depends on the subtypes of that type (e.g., for actor: agent, position and role).
Solution	Apply polymorphism on the model, applying e.g. hook strategy: – Declare an operation for that metric over the type of node which returns that value (hook version of the pattern) – Redefine the operation on those subtypes that assign a different value
Involved Clas-ses and Types	Parent: <<Node>> -- class whose metric' value is being computed Type: <<DataType>> -- the type of the metric
Assumptions	– Parent is root of a hierarchy, being $Heir_1$, …, $Heir_k$ its subclasses
Required knowledge	A total function assign which maps each subtype to the appropriate value, ass: <<Node>> → Type, such that domain(assign) = Parent's subclasses
Assumptions	∀j: 1 ≤ j ≤ k: $Heir_j$::metric is not defined explicitly ⇒ ass($Heir_j$) = Value
Form	– **context** Parent::metric(): Type **post**: result = Value – ∃i: 1 ≤ i ≤ k: [∀j: 1 ≤ j ≤ i: **context** $Heir_j$::metric(): Type **post**: result = *ass*($Heir_j$)]

Table 6. Example of pattern: the Dependency-Based pattern

Name	**Dependency-Based (Metrics Definition -> Quantitative -> Structural)**
Context	Some metrics have sense when applied to dependency links
Problem	The metrics will depend not just on the characteristics of the dependency link itself, but also on the two actors that act as depender and dependee
Solution	Identify three different factors that influence the metrics: one bound to the dependency link itself (probably related with the type of its dependum), and the others to the two actors, depender and dependee
Involved …	N/A
Assumptions	N/A
Required knowledge	– The effect of the depender, the dependee and the dependum in the metric, represented by three functions: filter: Dependum → Float correctionFactorDepender: Actor → Float, correctionFactorDependee: Actor → Float
Form	**context** DependencyLink::metric(): Type **let** ownerActor(x: DependableNode): Actor = **if** x.oclIsTypeOf(Actor) **then** x **else** x.owner **in**: **post**: result = self.dependency.dependum.filter() * ownerActor(self.dependency.depender).correctionFactorDepender() * ownerActor(self.dependee).correctionFactorDependee()

Table 7. Example of pattern: the Normalization pattern

Name	**Normalization (Metrics Transformation -> Numerical)**
Context	Some metrics may have a value that depend on the number of elements of a certain type that have influenced the metric
Problem	Often, the value of the metric should not depend on the number of elements processed
Solution	Normalize the value into some interval
Involved …	N/A
Assumptions	– The metric ranges onto a numerical data type – The normalization interval is [0.0, 1.0]
Required knowledge	– The value to be normalized, Value – The number of elements used to compute Value, Size
Form	**context** Element::metric(): Type **post**: Size = 0 **implies** result = 1.0 **post**: Size > 0 **implies** result = Value / Size

Table 8. Template for the Aggregation category

Name	**Aggregation (Metrics Definition -> Quantitative -> Structural)**
Context	A metric is defined over two types of different model elements such that elements of one type (aggregate) contain elements of the other (aggregee)
Problem	The value of a metric applied over the aggregate depends on the value applied over the aggregee
Solution	Define aggregate's metric as the combination of aggregee's
Involved …	Identical to Table 1
Assumptions	– Similar to Table 1
Required knowledge	– The relationship of aggregation from Aggregate to Aggregee, Aggregate::aggregees(): Set(Aggregee) – The combination function of aggregee's values into aggregate's, *aggregationFunction*

Table 9. Example of pattern specialization: from the Aggregation category into the Sum pattern

Name	**Sum**
Subtype-of	Aggregation[aggregationFunction ::= sum]
… (rest of the Sum pattern, see Table 1)	

specializes its supercategory by adding detail to some of the parts of the template. In this process, a partial instantiation is possible to bind some of the parameters of the supercategory pattern. As an example, in Table 8 we show an outline of the template for the Aggregation category and in Table 9 its specialization into the Sum pattern presented in Table 1. To make specialization explicit, we add a specialization clause in patterns. Eventually, we may get rid of some parts of the specialized template if there is redundancy.

6 Example of Application

In this section we show the applicability of the metrics pattern catalogue to one particular case, namely predictability as defined in [9].

Predictability is used in [23] as one of the properties of interest when analysing organizational styles. To obtain the metric as done in [9], we follow the following process (row's references in the text refer to Table 10):

- **Row 1.** From [23], we concluded that predictability is a measuring instrument for actors, therefore we declare the metric using the Individual Element and Measuring Instrument patterns. In [9], we defined actor predictability as the sum of the predictability of its stemming dependencies, including both the dependencies stemming from the actor itself and the dependencies stemming from the actor's intentional elements; to obtain these dependencies we apply a Navigation pattern. To obtain a normalized measure, between 0 and 1, we apply the normalization pattern to the value obtained from the sum described above. The process is described in detail in Fig. 2.

Table 10. Application of the metric definition process for Predicatibility

Row 1	See fig. 2, where this step is presented in detail and graphically

Row 2	Dependency::predictability = DiscriminationByType[Parent ::= Dependency; Type ::= Float; ass ::= dependencyTypePredictability]
	`context Dependency::predictability(): Float`
	`post: result = 1.0`
	`context GoalDependency::predictability(): Float`
	`post: result = self.goalPredictability()`
	`context SoftgoalDependency::predictability(): Float`
	`post: result = self.softgoalPredictability()`

Row 3	SoftgoalDependency::softgoalPredictability = Dependency-Based[filter ::= dependency.knowHow; correctionFactorDepender ::= dependerExpertise; correctionFactorDependee ::= 1] -- does not affect the result
	`context SoftGoal::softgoalPredictability(): Float`
	`let ownerActor(x: DependableNode): Actor = if x.oclIsTypeOf(Actor) then x else x.owner in`
	`post: result = ownerActor(self.dependency.depender).dependerExpertise() * self.dependency.dependum.knowHow()`

Row 4	Actor::dependerExpertise = ByCriterion[type ::= Float; criterion ::= true; judgement ::= Actor::expertise]
	`context Actor::dependerExpertise(): Float`
	`post: result = expertise(self.label)`

Row 5	`context Dependency::knowHow(): Float -- no pattern applied`
	`pre: self.type = Softgoal`
	`let theModel = Model.allInstances()->any() in:`
	`post: result = 1 - theModel.slope / contributionsToSoftgoalDep()+1`
	Dependency::contributionsToSoftgoalDep = Count[Aggregate ::= Dependency; Aggregee ::= DependeeLink; Type ::= Integer; aggregee ::= self.dependeeLink]
	`context Dependency::contributionsToSoftgoalDep(): Integer`
	`post: result = self.dependeeLink.contributionsToSoftgoalDep()->size()`
	DependeeLink::contributionsToSoftGoalDep = DiscriminationByTypeAndValue [Node ::= DependeeLink; type ::= Boolean; function ::= hasContributionLabel (as defined below)]
	`context DependeeLink::hasContributionLabel(): Boolean post: result = false`
	`context SoftgoalContribution::hasContributionLabel(): Boolean`
	`post: result = self.contr->notEmpty()`

Row 6	GoalDependency::goalPredicatibility = Inverse[Type ::= Integer; Value ::= self.nbTaskCombinations(); ResultIfZero ::= 0]
	GoalDependency::allTaskCombinations = TransitiveClousure [Elem :: = GoalDependency; Result ::= Set(Task); Expr ::= --not shown for the space reasons]
	`context GoalDependency::goalPredictability(): Float`
	`let nbTaskCombinations(x: GoalDependency): Integer = x.allTaskCombinations()->size() in`
	`post: self.nbTaskCombinations() = 0 implies result = 0`
	`post: self.nbTaskCombinations() > 0 implies result = 1 / self.nbTaskCombinations()`

- **Row 2.** Yu provides some rationale about the degree of freedom bound to dependencies [33, p. 15]. Analysing this rationale, we concluded that task and resource dependencies are totally predictable whilst goal and softgoal ones are not. Therefore, we define predictability of dependencies in terms of their type applying the Discrimination By Type pattern.
- **Row 3.** For softgoal dependencies, since softgoal satisfaction involves a compromise among depender and dependee, we use the Dependency-Based structural pattern, in which the depender side expresses the expertise of the depender actor to take informed decisions, whilst the dependee side measures the available know-how about that dependency (measured in terms of contributions to softgoals inside the dependee's SR). Since we need to refer to the owner actor of a dependency, that may be established in terms of the actor itself or some intentional element therein, we use a Navigational pattern to locate that actor.

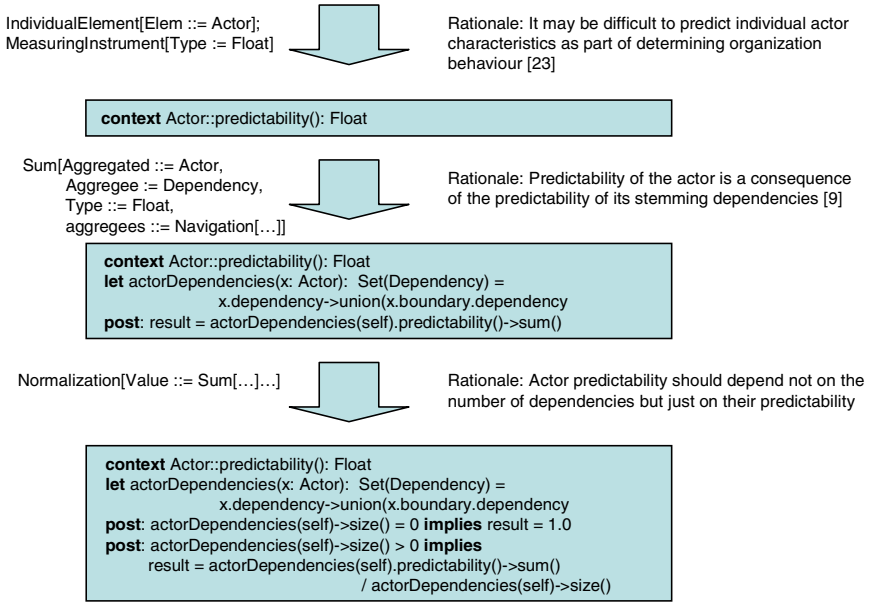

Fig. 2. Application of patterns in the first step of predictability definition

- **Row 4.** Concerning depender expertise, we considered that this knowledge cannot be computed from the model and then we apply a Qualitative pattern, namely By Criterion (of actor). Since the metric is applied to all actors, the criterion predicate must evaluate always to true.
- **Row 5.** Concerning know-how, we apply both the Count and Discrimination by Type and Value patterns to compute the number of dependees that state a contribution value to the dependum. In this case, however, we need to manipulate the result in order to create an inverse function as done in [9].
- **Row 6.** For goal dependencies, predictability was measured as the different ways of fulfilling the goal, generating all feasible task combinations using a Navigational pattern and then counting them (row 6). Then the Inverse Numerical pattern is applied to obtain the final result.

7 Conclusions and Future Work

In this paper we have motivated the need for having a catalogue of patterns for defining metrics over *i** models, proposed the general structure of such a catalogue, presented some patterns therein and illustrated their use with an example. Our framework facilitates the objective of analysing and comparing *i** models with respect some giving criteria in different contexts: business process reengineering, requirements validation, architecture assessment, etc.

We have validated this proposal using both our work and the related work. We comment here our own work. We have applied the framework retrospectively (as done in section 6) to 16 other metrics and obtained the following data: (i) 32 patterns of metrics definition (16 for Subject and 16 for Result); (ii) 44 for metrics definition, being Structural Discrimination patterns the most used by far (24 applications); (iii) 18 Numerical patterns applied, most of the times one for pattern as last manipulation to normalize the result; (iv) 11 Navigational patterns. It must be said that the metrics analyzed are not so complex as the Predictability studied in this paper. Also, we have checked with related work the applicability of the metrics with success. For instance, the metric proposed in [31] is a typical example also of applicability of Structural Discrimination patterns with qualitative assessment. This also happens with [22]. As an example, they define completeness as #{i∈InitialGoal| ∃ f∈FinalGoal·AllPositive(i,f)} / #InitialGoal. We may apply the sequence of patterns: Model+Classification Instrument for definition; Discrimination By Type and Value to apply the metric on goal satisfying the condition that are initial; Count for counting; Normalization to divide; Navigation to generate AllPositive. Similar for the others metrics in [22].

As a summary of this assessment, we may remark as fundamental characteristics of our approach the following:

- **Efficiency.** The process of defining metrics is greatly improved since the engineer needs just to identify the prototypical traits of the metrics and choose the appropriate patterns. The classification schema supports this selection. The existence of patterns aimed at capturing some time-consuming and cumbersome behaviour (numerical manipulation, navigational patterns) also supports efficiency.
- **Executability.** The use of OCL as metrics language allows using different types of tools, from OCL editors to validators and execution tools.
- **Expressivity.** Since the framework operates around the *i** conceptual data model, all the model elements may be considered. The addition of the concept of property allows defining not just structural metrics but also others more domain-oriented.
- **Robustness.** Using patterns, errors when defining the metrics are reduced once the patterns are validated. Numerical and navigational patterns are good examples of that. The use of assumptions in the pattern definition helps to establish explicitly which are the correctness conditions of the pattern.
- **Understandability and uniformity.** Using patterns, similar situations in different metrics are treated the same way, and the resulting metrics look similar, making easier their understanding.
- **Versatility.** The catalogue allows designing the metrics according to different concepts (the classification criteria): type of knowledge available, effort to invest, predominant model element, etc. Some of the patterns recognize the fact that it may be necessary to evaluate model elements in an individual basis, with some kind of qualitative judgement.

In relation to [7], we may say that our proposal helps in overcoming most of the drawbacks identified therein (at some degree, all, except modularity): refinement, because metrics can be defined at different levels of abstraction assessing the development process and helping in choosing between refinement alternatives; repeatability, because model similarity can be assessed by comparing values on appropriate metrics; complexity management, using the appropriate metrics (project-

oriented metrics) to drive model management; traceability, considered in terms of what parts of the model correspond to which domain concepts; reusability, because metrics can be used to decide if one model can be used in some context; scalability, because metrics help to analyse large models from several points of view; domain applicability, since not just the plain concepts of a domain may be represented but also metrics already defined for this domain.

As future work, we mention:

- To complete the catalogue with new, validated patterns and metrics constructed with them. As part of this goal, we aim at identifying other domains that may benefit from the existence of such catalogue, e.g. configuration management. Also, we plan to complement the catalogue with some classification schema to allow browsing the catalogue in a systematic way.
- To incorporate the catalogue into our current *i** modelling tools, REDEPEND-REACT for modelling component-based system architectures [17] and J-PR*i*M for driving business process reengineering processes [19]. Both tools currently allow defining structural metrics using some forms.
- To generalize the framework from *i** to a more general context. Since most of the concepts presented here are not particular of *i**, this is a feasible and logical goal to abstract the patterns into its metamodel level, in order to obtain a more generic form to be customized into a particular language and model. We think that this approach may be applied to the family of modelling languages structurally similar to *i** models, with a graph-oriented form, which includes several goal-oriented and other proposals. In this line of research, we plan to pay special attention to the analysis of completeness of the catalogues.
- To improve the definition of patterns by using metamodeling approaches to software metrics definition as those proposed in [3], [6]. We think that both approaches are complementary and would benefit from each other: patterns are methodologically-oriented whilst metamodeling is more foundational-oriented.

References

1. Ayala, C.P., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C.: A Comparative Analysis of i*-Based Goal-Oriented Modeling Languages. In: Proceedings 17th SEKE International Conference (2005)
2. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems 8(3) (2004)
3. Franch, X., Burgués, X., Ribó, J.M.: A MOF-Compliant Approach to Software Quality Modeling. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 176–191. Springer, Heidelberg (2005)
4. Briand, L., Morasca, S., Basili, V.R.: An Operational Process for Goal-Driven Definition of Measures. IEEE Transactions on Software Engineering 28(12) (2002)
5. Bryl, V., Giorgini, P., Mylopoulos, J.: Designing Cooperative IS: Exploring and Evaluating Alternatives. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 533–550. Springer, Heidelberg (2006)

6. Cachero, C., Calero, C., Poels, G.: Metamodeling the Quality of the Web Development Process' Intermediate Artifacts. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 74–89. Springer, Heidelberg (2007)
7. Estrada, H., Martínez, A., Rebollar, O., Pastor, J.: An Empirical Evaluation of the i* in a Model-Based Software Generation Environment. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, Springer, Heidelberg (2006)
8. Etien, A., Rolland, C., Salinesi, C.: Measuring the Business / System Alignment. In: Proceedings 1st REBNITA International Workshop (2005)
9. Franch, X.: On the Quantitative Analysis of Agent-Oriented Models. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 495–509. Springer, Heidelberg (2006)
10. Franch, X., Grau, G., Quer., C.: A Framework for the Definition of Metrics for Actor-Dependency Models. In: Proceedings 12th IEEE RE International Conference (2004)
11. Franch, X., Maiden, N.A.M.: Modeling Component Dependencies to Inform their Selection. In: Erdogmus, H., Weng, T. (eds.) ICCBSS 2003. LNCS, vol. 2580, pp. 81–91. Springer, Heidelberg (2003)
12. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and analizing early requirements in Tropos. Requirements Engineering Journal (REJ) 9(2) (2004)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
14. Genero, M., Miranda, D., Piattini, M.: Defining and Validating Metrics for UML Statechart Diagrams. In: Proceedings 5th ICEIS International Conference (2003)
15. Giorgini, P., Mylopoulos, J., Nicciarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, Springer, Heidelberg (2002)
16. Grau, G., Franch, X.: ReeF: Defining a Customizable Reengineering Framework. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 485–500. Springer, Heidelberg (2007)
17. Grau, G., Franch, X.: A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures. In: Oquendo, F. (ed.) ECSA 2007. LNCS, vol. 4758, pp. 139–155. Springer, Heidelberg (2007)
18. Grau, G., Franch, X.: Using the PRiM method to Evaluate Requirements Models with COSMIC-FFP. In: Proceedings MENSURA International Conference (2007)
19. Grau, G., Franch, X., Maiden, N.A.M.: PRiM: an i*-based process reengineering method for information systems specification. In: Information and Systems Technology (IST), vol. 50(1-2), Elsevier, Amsterdam (2008)
20. Guizzardi, R., Guizzardi, G., Perini, A., Mylopoulos, J.: Towards an Ontological Account of Agent-Oriented Goals. In: Choren, R., Garcia, A., Giese, H., Leung, H.-f., Lucena, C., Romanovsky, A. (eds.) SELMAS. LNCS, vol. 4408, pp. 148–164. Springer, Heidelberg (2007)
21. Jureta, I., Faulkner, S.: Tracing the Rationale Behind UML Model Change Through Argumentation. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, Springer, Heidelberg (2007)
22. Kaiya, H., Horai, H., Saeki, M.: AGORA: Attributed Goal-Oriented Requirements Analysis Method. In: Proceedings 10th IEEE RE International Conference (2002)
23. Kolp, M., Castro, J., Mylopoulos, J.: Organizational Patterns for Early Requirements Analysis. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, Springer, Heidelberg (2003)
24. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proceedings 5th ISRE International Symposium (2001)

25. Maiden, N.A.M., Robertson, S.: Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System. In: Proceedings 13th IEEE RE International Conference (2005)
26. Reijers, H.A., Vanderfeesten, I.T.P.: Cohesion and Coupling Metrics for Workflow Process Design. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 290–305. Springer, Heidelberg (2004)
27. Reynoso, L., Genero, M., Piattini, M., Manso, E.: Assessing the impact of Coupling on the Understandability and Modificaiblity of OCL expressions within UML/OCL combined models. In: Proceedings 11th METRICS International Symposium (2005)
28. Saeki, M.: Embedding Metrics into Information Systems Development Methods: An Application of Method Engineering Technique. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, Springer, Heidelberg (2003)
29. Sebastiani, R., Giorgini, P., Mylopoulos, J.: Simple and Minimum-Cost Satisfiability for Goal Models. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 20–35. Springer, Heidelberg (2004)
30. Susi, A., Perini, A., Mylopoulos, J., Giorgini, P.: The Tropos Metamodel and its Use. Informatica 29(4) (2005)
31. Sutcliffe, A., Minocha, S.: Linking Business Modelling to Socio-technical System Design. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626. Springer, Heidelberg (1999)
32. Yu, E., Mylopoulos, J.: Understanding Why in Software Process Modelling, Analysis, and Design. In: Proceedings 16th IEEE ICSE International Conference (1994)
33. Yu, E.: Modelling Strategic Relationships for Process Reengineering. PhD. thesis, University of Toronto (1995)