Evaluating Goal Models within the Goal-oriented Requirement Language

Daniel Amyot[1], Sepideh Ghanavati[1], Jennifer Horkoff[2], Gunter Mussbacher[1], Liam Peyton[1] and Eric Yu[3]

[1] *SITE, University of Ottawa, { damyot | sghanava | gunterm | lpeyton }@site.uottawa.ca*
[2] *Department of Computer Science, University of Toronto, jenhork@cs.utoronto.ca*
[3] *Faculty of Information, University of Toronto, yu@fis.utoronto.ca*

Abstract

In this paper, we introduce the application of rigorous analysis procedures to goal models in order to provide several benefits beyond the initial act of modelling. Such analysis can allow modellers to assess the satisfaction of goals, facilitate evaluation of high-level design alternatives, help analysts decide on the high-level requirements and design of the system, test the sanity of a model, and support communication and learning. The analysis of goal models can be done in very different ways depending on the nature of the model and the purpose of the analysis. In our work, we use the *Goal-oriented Requirement Language* (GRL), which is part of the *User Requirements Notation* (URN). URN, a new Recommendation of the International Telecommunications Union, provides the first standard goal-oriented language. Using GRL, we develop an approach to analysis that can be done by evaluating qualitative or quantitative satisfaction levels of the actors and intentional elements (e.g., goals and tasks) composing the model. Initial satisfaction levels for some of the intentional elements are provided in a *strategy* and then propagated to the other intentional elements of the model through the various links that connect them. The results allow for an assessment of the relative effectiveness of design alternatives at the requirements level. Although no specific propagation algorithm is imposed in the URN standard, different criteria for defining evaluation mechanisms are described. We provide three algorithms (quantitative, qualitative, and hybrid) as examples, which satisfy the constraints imposed by the standard. These algorithms have been implemented in the open-source jUCMNav tool, an Eclipse-based editor for URN models. The algorithms are presented and compared with the help of a telecommunication system example.

1. Introduction

Requirements engineering is concerned with the elicitation, analysis, specification, validation, and management of requirements. A variety of approaches have been taken to modelling requirements since the 1970s from conceptual entity-relationship modelling, to structured, object-oriented, use case and goal-oriented approaches. A good summary of these approaches is given by Mylopoulos, Chung and Yu,[1] with the observation that goal-oriented requirements engineering (GORE) is generally complementary to other approaches. It is well suited to analyzing requirements early in the software development cycle, especially with respect to non-functional requirements and the evaluation of alternatives whereas other approaches tend to focus more on analyzing requirements later in the software development cycle with a focus on traceability between requirements and implementation.

GORE is an approach used to uncover, analyze, and describe stakeholder goals leading to software and system requirements. This popular approach, supported by a large international community for more than 15 years, has proven successful in assisting requirements and software engineers in their activities by enabling novel types of analyses over non-functional properties of systems and facilitating the documentation of design rationale.[2] Numerous languages and notations have been developed over the years to model goals and their relationships in an explicit way. For instance, popular languages include KAOS,[3,4] the NFR Framework,[5] *i**,[6,7] and TROPOS.[8] More recently, the *Goal-oriented Requirement Language* (GRL) has become an internationally recognized standard for goal-oriented modelling, as part of a new Recommendation of the International Telecommunications Union named *User Requirements Notation* (URN).[9–11]

The application of rigorous analysis procedures to goal models can provide several benefits beyond the initial act of modelling. Such analysis can allow modellers to assess the satisfaction of goals, facilitate evaluation of high-level design alternatives, help analysts decide on the high-level requirements and design of the system, test the sanity of a model, and support communication and learning. Many existing attempts to analyze goal-oriented models[5,8,12–14] demonstrate that the analysis of goal models can be done in very different ways depending on the nature of the model and the purpose of the analysis.

In our work, the goal models we analyze are described using GRL. GRL integrates the core concepts of *i**[7] and the NFR Framework[5] and supports links to a companion scenario notation in URN called *Use Case Maps* (UCM). Using GRL, we develop an approach to analysis that can be done by evaluating qualitative or quantitative satisfaction levels of the

intentional elements (e.g., goals and tasks) and actors composing the model. Initial satisfaction levels for some of the intentional elements are provided in a *strategy* and then propagated to the other intentional elements of the model through the various links that connect them. Results are examined to determine the effectiveness of the high-level requirements contained within the strategy. Although no specific propagation algorithm is imposed in the Recommendation of the International Telecommunications Union, different criteria for defining evaluation mechanisms are described.

We provide three algorithms (quantitative, qualitative, and hybrid) as examples, which satisfy the constraints imposed by the standard. These algorithms have been implemented in the open-source jUCMNav tool, an Eclipse-based editor for URN models.[15] The algorithms are presented and compared with the help of a telecommunication system example.

An overview of the Goal-oriented Requirement Language[†] is given in Section 2 to better understand the concepts used in evaluating strategies. We also introduce an example goal model that we use throughout the rest of the paper. In Section 3, we define a list of criteria that serves to categorize many types of algorithms used for the evaluation of goal models. Three specific algorithms for analyzing GRL models are introduced in Section 4 and then applied to our telecommunication example in Section 5. Related work is discussed in Section 6, followed by our conclusions in Section 7.

2. Goal-oriented Requirement Language

2.1 Overview of URN

URN allows software and requirements engineers to discover and specify requirements for a proposed or an evolving system, and analyse such requirements for correctness and completeness.[11] URN combines the *Goal-oriented Requirement Language* (GRL) for

[†] The definition of GRL, the criteria for characterizing analysis algorithms, and the three specific algorithms presented in this paper were developed by us over the years and are now included in ITU-T's Recommendation Z.151 respectively in clause 7, clause 11.1, and Appendix II.[11] Earlier contributors to the definition of GRL included Lin Liu, Daniel Gross, and Luiz Marcio Cysneiros. Jean-François Roy contributed to the concepts of GRL strategies and actor evaluation, along with prototype tool support. The readers are referred to Recommendation Z.151 for the full and authoritative definition of URN, including GRL.

modelling goal-oriented and intentional concepts (related to non-functional requirements, quality attributes, and reasoning about alternatives) with the *Use Case Map* (UCM) notation for modelling scenario concepts (related to operational requirements, functional requirements, and performance and architectural reasoning). In particular, URN has concepts for the specification of stakeholders, goals, non-functional requirements, rationales, behaviour, scenarios, scenario participants, and high-level architectural structure. An extensive body of research comprising over 220 publications and theses related to URN, GRL and UCM, is available at the URN Virtual Library.[16]

2.2 Overview of Goal-oriented Requirement Language

Major benefits of GRL over other popular goal notations include the integration of GRL with a scenario notation, the support for qualitative and quantitative attributes, and a clear separation of GRL model elements from their graphical representation, enabling a scalable and consistent representation of multiple views/diagrams of the same goal model.

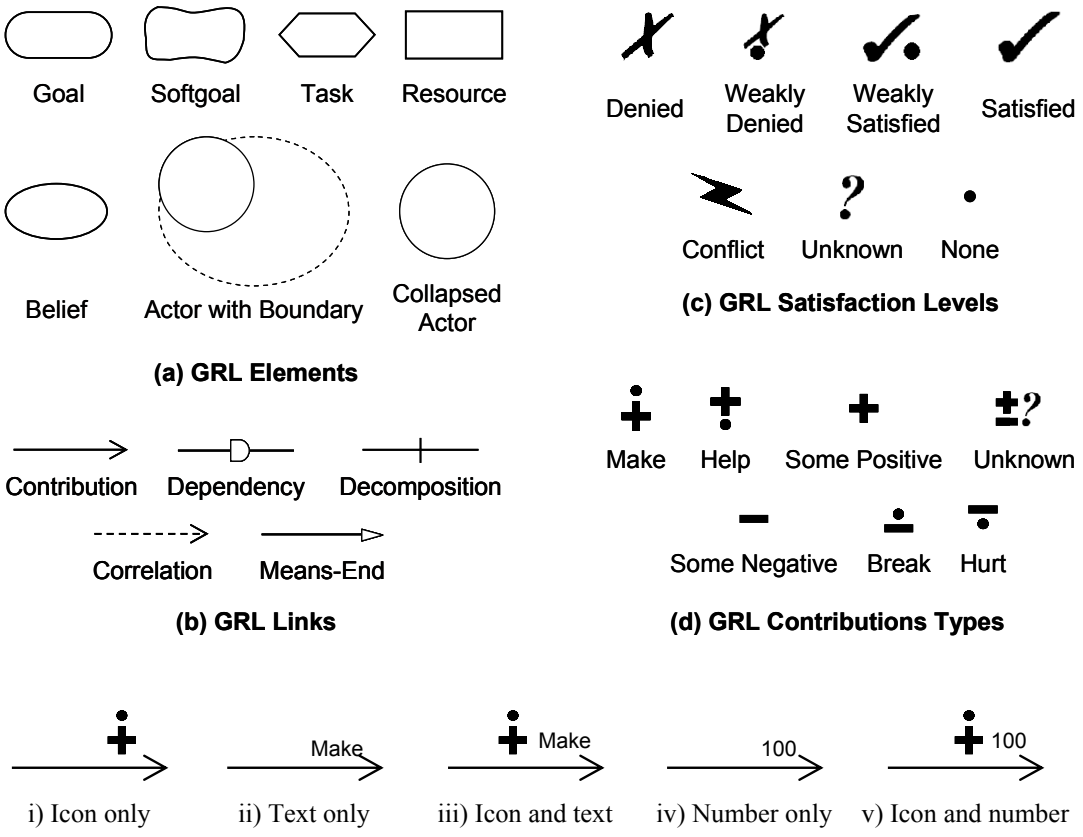


Figure 1 Basic elements of the GRL notation

The syntax of GRL (see Figure 1) is based on the syntax of the *i** language.[7] A GRL diagram shows the high-level business goals and non-functional requirements of interest to a stakeholder and the alternatives for achieving these goals and requirements. A goal diagram also documents *beliefs* (facts) important to the stakeholder.

In addition to beliefs, *GRL intentional elements* (see Figure 1a) can be softgoals, goals, tasks, and resources. *Softgoals* differ from *goals* in that there is no clear, objective measure of satisfaction for a softgoal whereas a goal is quantifiable. In general, softgoals are related to non-functional requirements, whereas goals are related to functional requirements. *Tasks* represent solutions to (or operationalizations of) goals or softgoals. In order to be achieved or completed, softgoals, goals, and tasks may require *resources* to be available. An *actor* represents a stakeholder of the system or another system. Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available and softgoals to be satisfied.

GRL element links (see Figure 1b) are used to connect isolated elements in the requirement model using structural and intentional relationships (including decompositions, contributions, and dependencies). *Decomposition links* allow an element to be decomposed into sub-elements. AND, IOR, as well as XOR decompositions are supported. XOR and IOR decomposition links may alternatively be displayed as *means-end* links. *Contribution links* indicate desired impacts of one element on another element. A contribution link can have a qualitative contribution type (see Figure 1d), or a quantitative contribution (integer value between -100 and 100, see Figure 1e). *Correlation links* are similar to contribution links, but describe side effects rather than desired impacts. Finally, *dependency links* model relationships between actors (one actor depending on another actor for something).

Similar to the NFR framework,[5] GRL provides support for evaluations to analyze the most appropriate trade-offs among (often conflicting) goals of stakeholders. In GRL, the starting point of an evaluation is embodied in a *strategy*, which consists of a set of intentional elements that are given initial satisfaction values. These satisfaction values, which can be qualitative (see Figure 1c) or quantitative, capture contextual or future situations as well as choices among alternative means of reaching various goals. These values are then propagated to the other intentional elements through their links taking contribution types into account. This enables a global assessment of the strategy being studied. An *importance* attribute (again quantitative or qualitative) may also be specified for intentional elements inside actors, which is used when evaluating strategies for the goal model. The *importance* is shown between

parentheses in intentional elements: (H)igh, (M)edium, (L)ow, or None for qualitative evaluations, and an integer between (1) and (100) for quantitative evaluations (None and 0 are not displayed on diagrams).

2.3 GRL Example

Figure 2 shows an example GRL diagram (adapted from previous work[9]) that describes the impact of the selection of the location of a new wireless service and its data in an existing network. This example shows that intentional elements can be decomposed, and that many local alternatives can have various impacts on different concerns of the stakeholders involved, with no obvious global solution that would satisfy everyone. Different strategies and different evaluation algorithms will be explored for this example in Section 5.

For the service provider, keeping a low cost is important but providing high performance is even more important. Two alternative solutions are considered for the data location: locate it in an existing service control point (SCP) or in a new service node (SN). The latter would require changes to the existing infrastructure of the service provider, and it depends on the capacity of a vendor to provide service nodes at the appropriate time. Two alternatives also exist for the location of the service logic: in the central switch itself or in an external service control point. The first option requires fewer message exchanges on the network but increases the load on the switch.

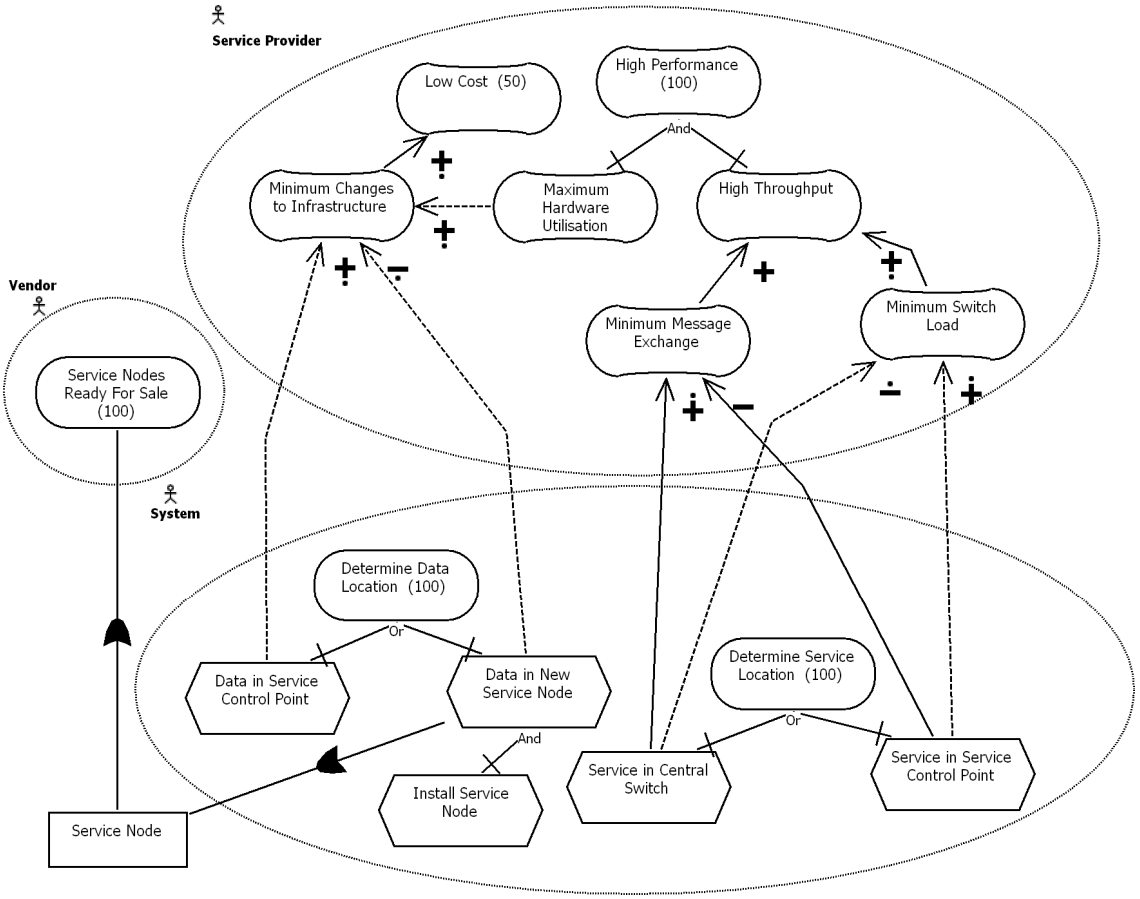


Figure 2 GRL model example

2.4 GRL Abstract Metamodel

The specification of URN complies with the language requirements described in Recommendation Z.150,[10] and it adheres to the guidelines of the Recommendation Z.111 for metamodel-based definitions of ITU-T languages.[17] Recommendation Z.151 specifies the abstract metamodel of URN (discussed here as it relates to GRL), a concrete graphical syntax for URN (discussed in Section 2.2 for GRL), an XML-based interchange format for URN, and a data language for conditions and expressions in URN.

Figure 3 describes the part of the abstract URN metamodel specific to GRL, which formalizes the concepts introduced in section 2.2. The diagram shows the relationship of *GRLLinkableElement*s (i.e., actors and intentional elements) with *ElementLink*s (i.e., decomposition, dependency, and contribution links). On the right hand side of the diagram, the concepts required for the evaluation of GRL models (*StrategiesGroup*, *EvaluationStrategy*, and *Evaluation*) are defined.

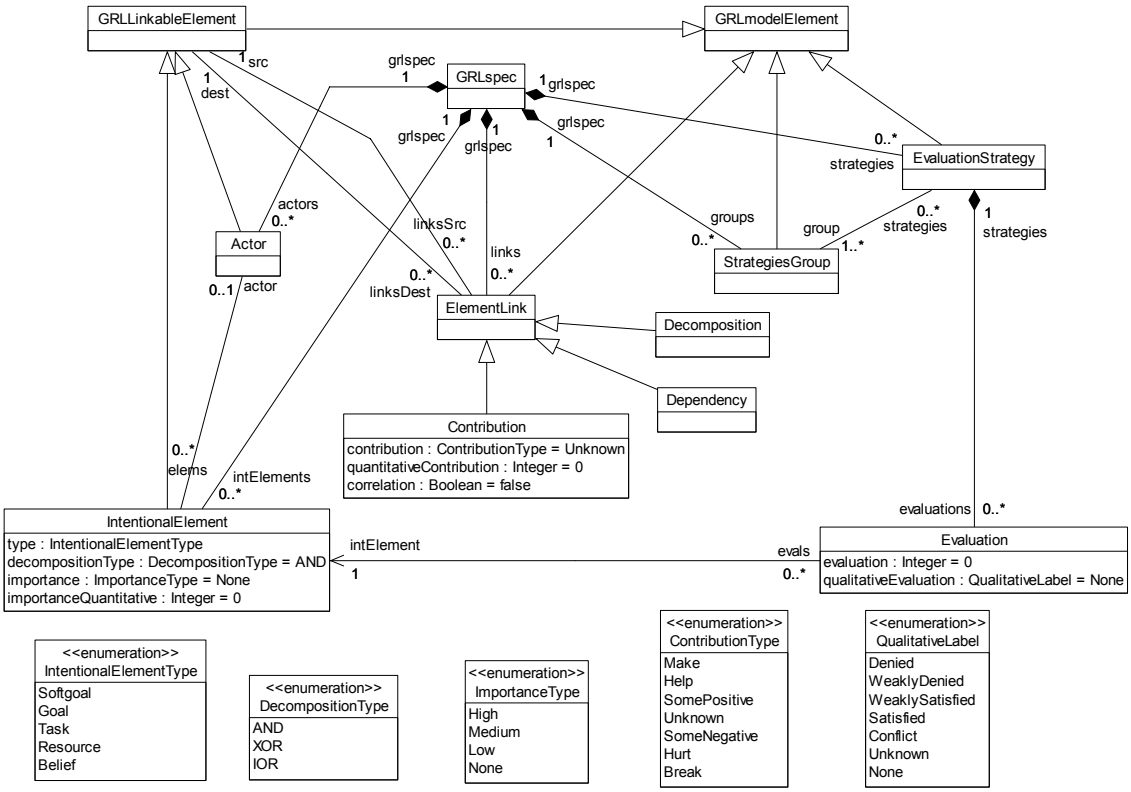


Figure 3 Abstract grammar: GRL

In Recommendation Z.151, the static semantics of GRL is defined with the help of natural language descriptions and constraints on the abstract metamodel. The dynamic aspects of GRL are defined by requirements and guidelines for *propagation mechanisms* for GRL model evaluation. The GRL model evaluation allows for the comparison of alternatives and facilitates trade-offs among conflicting goals of various stakeholders. The standard does not enforce a specific propagation mechanism as GRL can be used in different ways by different modellers, e.g. for qualitative evaluations or quantitative ones.

2.5 Tool Support with jUCMNav

The best tool supporting URN modelling, analysis, and transformations is the open-source Eclipse plug-in *jUCMNav*.[15] It supports analysis features for evaluating GRL models (see Figure 4) and executing UCM models. A URN model can contain multiple interlinked GRL and UCM diagrams, and with definitions of actors, intentional elements, intentional links, responsibilities and components shared across diagrams. jUCMNav prevents the creation of syntactically incorrect URN models and supports the verification of user-defined static semantic rules written in OCL. The tool also supports exports to various bitmap and report

formats, the import/export of GRL catalogues (partial models), and the export of results of GRL strategy evaluations in a comma-separated value file.

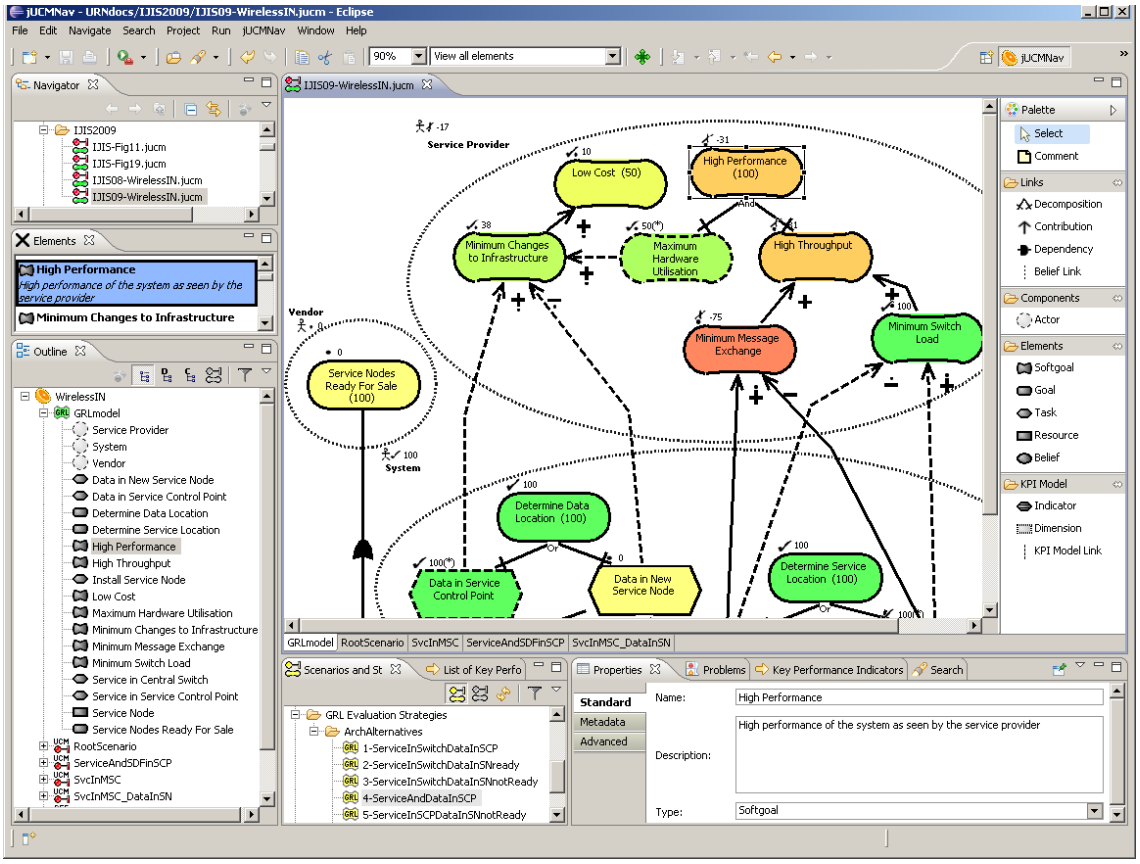


Figure 4 jUCMNav tool: GRL editor with strategy evaluation

3. Adding Support for GRL Evaluation Algorithms

Goal models can be analysed in many ways, depending on the purpose of these models.[3,4,13] During the development of the URN standard, it was considered premature to attempt to standardize evaluation algorithms, but there is support for modellers to define their own algorithms.

In our work (which has been integrated into the jUCMNav tool), a GRL model is evaluated by assigning satisfaction values to a subset of intentional elements and then propagating these values to the other intentional elements via decomposition, contribution and dependency links. Each GRL model can be evaluated several times with different subsets of intentional elements (alternatives) through assigning different evaluation strategies. During GRL model evaluation, for each GRL *IntentionalElement* and each GRL Actor, two evaluation values can be selected: one that is qualitative (qualitativeVal of type *QualitativeLabel*) and one that is quantitative (quantitativeVal, an integer value in the range [-

100..100]). Each evaluation strategy has its value (quantitative or qualitative) initialized to its associated *IntentionalElement*. The quantitativeVal and qualitativeVal evaluation attributes of all other intentional elements and of all actors are initially set to 0 and None respectively.

From this initial context, goal models can be evaluated in many ways. Algorithms and tools for GRL model evaluation should consider the following non-exhaustive list of criteria when describing and formalizing evaluation algorithms. Many of these criteria are actually independent from GRL and can be applied to classify analysis approaches for other goal modelling languages.

a) **Evaluation type**: Three general types of evaluations are considered:

- *Quantitative evaluation*: uses the quantitativeContribution attribute of *Contribution*, the importanceQuantitative attribute of *IntentionalElement*s, and the quantitativeVal attribute of intentional elements initialized from the selected *EvaluationStrategy*. This type of evaluation may be the most appropriate when specific measures from the domain can be added to models.

- *Qualitative evaluation*: uses the qualitative contribution attribute of *Contribution*, the importance attribute of *IntentionalElement*s, and the attribute qualitativeVal of intentional elements initialized from the selected *EvaluationStrategy*. This type of evaluation may be most appropriate when specific, quantitative data from the domain is difficult to acquire.

- *Hybrid evaluation*: uses another combination of the above three categories of attributes (for contribution, importance, and intentional element evaluation value). This type of evaluation may be appropriate when only partial quantitative domain measures are available.

b) **Propagation direction**: Three different propagation directions for the evaluation values through different GRL element links are considered:

- *Forward propagation*: the *EvaluationStrategy* initializes some of the *IntentionalElement*s in the GRL model (source nodes, often leaves in the graph), and the evaluation values are propagated in a bottom-up way to higher-level intentional elements (targets) of the model. The results of selected alternatives can be analysed and conflicts detected. This type of evaluation helps to answer "What if?" questions.

- *Backward propagation*: the *EvaluationStrategy* initializes some of the *IntentionalElement*s in the GRL model (target nodes, often roots in the graph), and the evaluation values are propagated in a top-down way to lower-level intentional elements (sources) of the model. Such propagation is used to find a set of alternatives that, if satisfied, would lead to the initial values provided, helping to answer "Is it possible?" questions.

- *Mixed propagation*: a combination of the above where the *EvaluationStrategy* initializes some of the *IntentionalElement*s that are neither leaves nor roots. From these elements, forward propagation is used to compute evaluation values of higher-level intentional elements whereas backward propagation is used to compute evaluation values of lower-level intentional elements.

c) **Actor satisfaction**: An algorithm may calculate actor evaluation levels, or not.

d) **Automation**: An algorithm may be fully automated, or interactive (e.g., to resolve conflicts).

e) **Cycles**: An algorithm may handle cycles in GRL models (completely or partially), or require models to be acyclic.

f) **Conflicts**: An algorithm may determine that multiple contributions targeting the same intentional elements are conflicting, or not. In addition, if conflicts are detected, then there could be one or many categories of conflicts.

g) **Strategy consistency**: An algorithm may allow inconsistent strategies, or not. An *EvaluationStrategy* is inconsistent if some of the initial evaluations it contains propagate into evaluations of intentional elements that are also initialized by the strategy, but with different values.

h) **Evaluation overriding**: An algorithm may allow some evaluations defined as part of a strategy to be overridden during the propagation, or not.

i) **Relation to UCM**: The results of the propagation may impact the values of UCM scenario variables, or not. In addition, updates to UCM scenario variables after path traversal may impact intentional element evaluations, or not.

j) **Evaluation ordering for links**: GRL element links (decompositions, contributions, and dependencies) may be evaluated in different orders. An algorithm should either specify that order or mention that there is no order.

k) **Link evaluations**: An evaluation algorithm should provide functions to compute results of decomposition, contribution, and dependency link usages.

l) **Tolerance**: For contribution links, an algorithm may define a tolerance to help decide whether an intentional element becomes satisfied or just weakly satisfied (and respectively denied or just weakly denied) because of contributions. A non-null tolerance prevents that many partial satisfaction or denial values from contributions result in a fully satisfied or denied element.

Other criteria for which GRL could not easily offer support (e.g., probabilistic evaluations and separate values for satisfaction and denial) are out of scope for this paper.

GRL model evaluations are performed using the language's abstract syntax, independently of the presence or absence of GRL diagrams. During model evaluations however, the presentation of intentional elements and actors in GRL diagrams is updated to reflect the current evaluation values. This is done with qualitative satisfaction labels (defined in Figure 1c) or quantitatively with integer values, depending on the type of analysis. Initial values set by a strategy are further indicated with a star (*) on the intentional elements of GRL diagrams (and jUCMNav further displays these intentional elements with dashed lines).

4. Algorithms for GRL Model Evaluation

4.1 Overview and Characteristics

This section illustrates three algorithms that are defined for GRL model evaluation. These algorithms share the following common characteristics, explained in Section 3: (b) Forward propagation; (c) Actor satisfaction is evaluated; (d) Fully automated; (e) Cycles in models are handled partially: a cycle will only be evaluated if one of its elements has a value initialized by the strategy; (g) Inconsistent evaluation strategies are allowed; (h) Evaluations defined as part of a strategy are not overridden; and (j) Element links are evaluated in the following order: decompositions, contributions, and dependencies. A generic algorithm based on the above characteristics is presented in Section 4.2. The differences between these three algorithms can be summarized as follows:

- Section 4.3: (a) Quantitative evaluation, (f) no conflict detection, (i) with relation to UCM, and (l) with tolerance.
- Section 4.4: (a) Qualitative evaluation, (f) conflict detection, (i) without relation to UCM, and (l) without tolerance.
- Section 4.5: (a) Hybrid evaluation, (f) no conflict detection, (i) with relation to UCM, and (l) with tolerance.

As for link evaluation functions (k), they will be explained in detail for each algorithm. Algorithms are explained in plain text when they are trivial, and with pseudo-code that takes advantage of the GRL abstract metamodel (Figure 3) when they are not trivial.

4.2 Generic Algorithm Overview

Our algorithms all follow the same three steps: 1) initialize the evaluation values of the GRL intentional elements based on the strategy selected, 2) do a forward propagation of the evaluation values to the other elements, and 3) calculate the satisfaction of actors. The first step follows the requirements presented in Section 3, and the third step depends on the type of evaluation chosen. This section discusses the second step in more details, as it is common to the three example evaluation algorithms illustrated here.

The forward propagation algorithm in Figure 5 follows a bottom-up, automated approach that can handle cycles partially and that does not override the initial evaluation values provided by a strategy, even when inconsistent. This algorithm takes as inputs the GRL specification and the selected strategy. It outputs a hash map containing a new evaluation value for each intentional element. In this algorithm, each intentional element knows its number of incoming source links (*totalSourceLink*) and tracks the number of links that have been used in the propagation so far (*linkReady*).

The forward propagation algorithm invokes the *calculateEvaluation* algorithm (Figure 6), which first checks whether the element is initialized by the strategy, and if necessary computes a value from decomposition links (*CalculateDecompositions*), then considers contribution links (*CalculateContributions*), and then dependencies (*CalculateDependencies*). The result of *CalculateDecompositions* is an input for *CalculateContributions*, and the result of *CalculateContributions* is an input for *CalculateDependencies*. The result of *CalculateDependencies* is the final evaluation value.

The *EvaluationValue* type here is a placeholder for the type of evaluation (*QualitativeLabel* for qualitative evaluations, and *Integer* for quantitative evaluations). The content of the three sub-algorithms invoked here depends on the general type of evaluation and will be detailed for each approach.

```
Algorithm ForwardPropagation
Inputs GRLmodel:GRLspec, currentStrategy:EvaluationStrategy
Output newEvaluations:HashMap

elementsReady:List = ∅          // intentional elements that can be evaluated
elementsWaiting:List = ∅        // intentional elements that cannot yet be evaluated

newEvaluations = ∅

for each element:IntentionalElement in GRLmodel.intElements
{
        element.linkReady = 0
        if (element in currentStrategy.evaluations.intElement) // is the element initialized?
            elementsReady.add(element)
        else
            elementsWaiting.add(element)
}

while (elementsReady.size() > 0)
{
        element = elementsReady.get()
        elementsReady.remove(element)
        newEvaluations.add(element, CalculateEvaluation(element, currentStrategy))
        for each link:ElementLink in element.linksSrc
        {
            destination = link.dest
            destination.linkReady = destination.linkReady + 1
            if (destination.linkReady == destination.totalSourceLink)
            {
                // all source elements have known evaluation values
                elementsWaiting.remove(destination)
                elementsReady.add(destination)
            }
        }
}

return newEvaluations
```

Figure 5 Example: Forward propagation algorithm

```
Algorithm CalculateEvaluation
Inputs element:IntentionalElement, currentStrategy:EvaluationStrategy
Output satisfactionValue:EvaluationValue

decompValue:EvaluationValue       // intermediate result
contribValue:EvaluationValue      // intermediate result

if not(element in currentStrategy.evaluations.intElement) // is the element not initialized?
{
        // calculate based on decompositions, contributions, and dependencies
        decompValue = CalculateDecompositions(element)
        contribValue = CalculateContributions(element, decompValue)
        satisfactionValue = CalculateDependencies(element, contribValue)
}

return satisfactionValue
```

Figure 6 Example: Calculate evaluation algorithm

4.3 A Quantitative Evaluation Algorithm

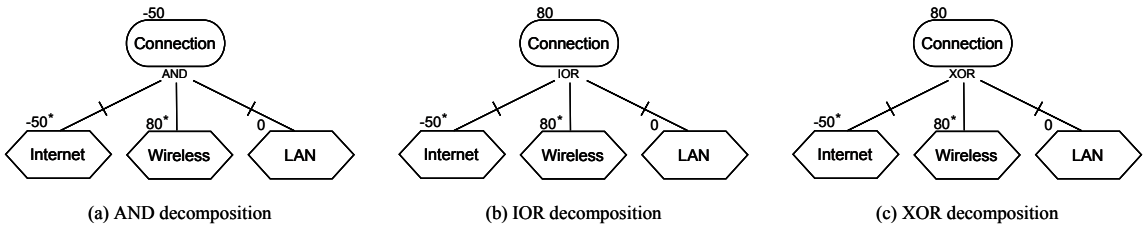
This quantitative GRL algorithm uses *Integer* values for the evaluation, and hence uses the quantitativeContribution attribute of *Contribution*, the importanceQuantitative attribute of *IntentionalElement*s, and the new quantitativeVal attribute of intentional elements initialized from the selected *EvaluationStrategy*. These quantitative values can range from -100 to 100. As mentioned earlier, first the algorithm calculates the evaluation values for decomposition links, then contribution links and finally dependency links.

4.3.1 Calculating quantitative evaluations for decomposition links

This corresponds to the CalculateDecompositions(element) step in Figure 6. The result depends on the type of decomposition (AND, IOR, or XOR).

The satisfaction level of an intentional element with an AND-type decomposition link is the *minimum* value of the quantitative evaluation values of its source elements. For an IOR-type decomposition link, the satisfaction level is the *maximum* value of the quantitative evaluation of its source elements. For an XOR-type decomposition link, the *maximum* is also used, but a warning is generated if more than one source element has a quantitative evaluation value different from 0.

Figure 7 provides an example of each decomposition type based on a strategy where two sources out of three are initialized (*). The difference between (b) and (c) here is that evaluating (c) will generate a warning as two sources have values different from 0.



(a) AND decomposition (b) IOR decomposition (c) XOR decomposition

Figure 7 Example: Quantitative evaluation of decomposition links

4.3.2 Calculating quantitative evaluations for contribution links

This corresponds to the CalculateContributions(element, decompValue) step in Figure 6. The total quantitative contribution is the sum of the products of the quantitative evaluation of each source element by its quantitative contribution level to the element. This value is added to decompValue up to a predefined tolerance if there is no fully satisfied or denied contribution (i.e., with the help of the tolerance, it is avoided that many partial satisfaction or denial values result in a fully satisfied or denied element; e.g., 20 satisfaction values of 5 each are not 100 but 100 minus tolerance). Correlations are treated the same way as contributions.

```
Algorithm CalculateContributions
Inputs element:IntentionalElement, decompValue:Integer
Output contribValue:Integer

tolerance:Integer        // predefined tolerance, between 0 and 49
oneCont:Integer          // one weighted contribution
totalCont:Integer = 0    // weighted sum of the contribution links
hasSatisfy:Boolean       // a weighted contribution of 100 is present
hasDeny:Boolean          // a weighted contribution of -100 is present
hasSatisfy = (decompValue == 100)
hasDeny = (decompValue == -100)

// compute the weighted sum of contributions
for each link:Contribution in element.linksDest
{
        oneCont = link.src.quantitativeVal × link.quantitativeContribution
        totalCont = totalCont + oneCont
        if (oneCont == 100) hasSatisfy = true
        if (oneCont == -100) hasDeny = true
}
totalCont = totalCont / 100
contribValue = totalCont + decompValue
```

```
// contribution value cannot be outside [-100..100]
if (|contribValue| > 100)
        contribValue = 100 × (contribValue/|contribValue|)

// take tolerance into account if a weighted contribution of 100 or -100 is not present
if ((contribValue ≥ 100 – tolerance) and not(hasSatisfy))
        if (totalCont > 0) // positive contribution
                contribValue = max (decompValue, 100 – tolerance)        // case A
                // else there is nothing to do, contribValue remains unchanged.
else if ((contribValue ≤ -100 + tolerance) and not(hasDeny))
        if (totalCont < 0) // negative contribution
                contribValue = min (decompValue, -100 + tolerance)       // case B
                // else there is nothing to do, contribValue remains unchanged.
return contribValue
```

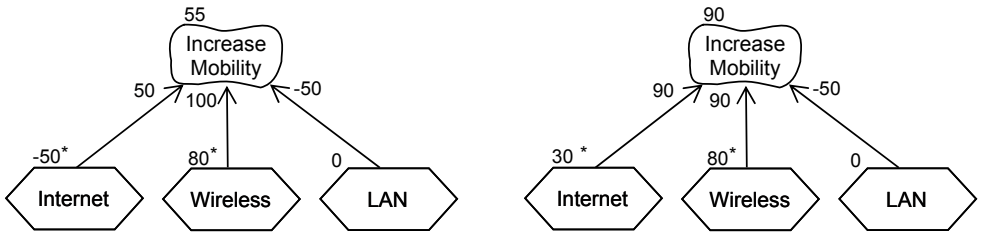
Figure 8 Example: Quantitative CalculateContributions algorithm

The algorithm in Figure 8 ensures that the satisfaction level of each intentional element will not go above 100 or below -100. In addition, the algorithm takes a *tolerance* into account to ensure that the evaluation value of an intentional element can be 100 (respectively -100) only if a) at least one of the intentional elements that contribute to the element has a weighted contribution of 100 (respectively -100) or b) decompValue is 100 (respectively -100). If this is not the case, then the evaluation value may be adjusted as specified in Figure 8 and illustrated for positive values in Table 1 (negative values are handled analogously). Note that this quantitative propagation algorithm resolves conflicts automatically by summing negative and positive evidence to produce a single value. In choosing an evaluation algorithm, the user should consider whether this way of resolving conflicts is appropriate for the domain in question.

Table 1 Example: Calculating contribution values with different tolerance values

Case in Figure 8	*has Satisfy*	*decomp Value*	*total Cont*	*tolerance*	*tolerance limit*	*contribValue*
A	FALSE	95	3	10	100 - 10 = 90	max (decompValue, 90) = **95**
hasSatisfy	TRUE	95	3	10	100 - 10 = 90	95 + 3 = **98**
below tolerance limit	FALSE	95	3	1	100 - 1 = 99	95 + 3 = **98**
totalCont is negative	FALSE	95	-3	10	100 - 10 = 90	92 - 3 = **92**
hasSatisfy	TRUE	95	-3	10	100 - 10 = 90	95 - 3 = **92**
below tolerance limit	FALSE	95	-3	4	100 - 4 = 96	95 - 3 = **92**
A	FALSE	85	13	10	100 - 10 = 90	max (decompValue, 90) = **90**
hasSatisfy	TRUE	85	13	10	100 - 10 = 90	85 + 13 = **98**
below tolerance limit	FALSE	85	13	8	100 - 1 = 99	85 + 13 = **98**

Figure 9 provides two examples with three contributions each (the initial decompValue is 0). Strategies initialize two elements. In (a), we have (-50 × 50) + (80 × 100) + (0 × -50) = 55. In (b), where the tolerance has been set to 10, we have (30 × 90) + (80 × 90) + (0 × -50) = 99. However, in the second case, as there is no fully satisfied weighted contribution and decompValue is not 100, then 100-tolerance = 90 is output.



(a) Contributions (b) Contributions with a tolerance of 10

Figure 9 Example: Quantitative evaluation of contribution links

4.3.3 Calculating quantitative evaluations for dependency links

This corresponds to the CalculateDependencies(element, contribValue) step in Figure 6. In this algorithm, the source element of the dependency links cannot have an evaluation value higher than those of the intentional elements it depends on (i.e., the target elements of the dependency links). This algorithm hence simply returns the minimum between contribValue and the evaluation values of the target elements.

A simple example is shown in Figure 10, with a strategy that initializes the two tasks. Consequently, the qualitative values of other elements are initially set to 0. Internet Connection becomes -75 since this value is less than 0. Low Costs, on the other hand, will keep its value of 0 because it is less than 50. The Increase Visibility softgoal gets the value min(0, min(-75, 0)) = -75.

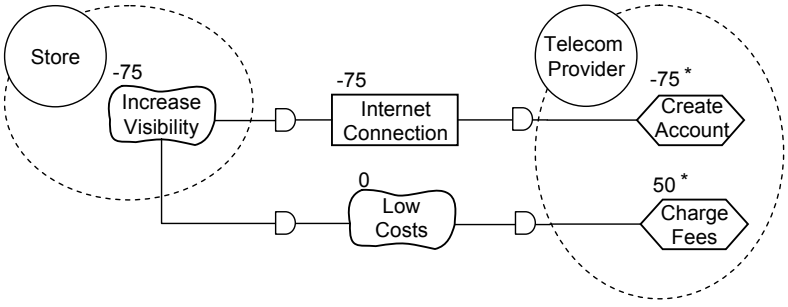


Figure 10 Example: Quantitative evaluation of dependency links

4.3.4 Calculating quantitative evaluations for actors

This is the third and last step discussed in Section 4.2. In order to compute the quantitative evaluation value of an actor, it is necessary to identify the quantitative satisfaction value and quantitative importance value of each intentional element bound to the actor. Only elements with an importance greater than 0 are counted (assume their number to be n and their references to be $elem_i$ with $i = 1..n$). This algorithm then computes the quantitative evaluation value of the actor as follows:

$$actor.quantitativeVal = \frac{\sum_{i=1}^{n} elem_i.quantitativeVal \times elem_i.importanceQuantitative}{\sum_{i=1}^{n} elem_i.importanceQuantitative}$$

For example, Figure 11 shows an actor with four softgoals, three of which with non-zero importance. The quantitative value of the actor's satisfaction becomes:

$$((100 \times 100) + (100 \times 29) + (-75 \times 60)) / (100 + 19 + 60) = 44$$

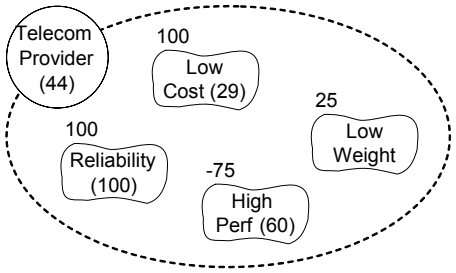


Figure 11 Example: Quantitative evaluation of actors

4.4 A Qualitative Evaluation Algorithm

Our qualitative GRL algorithm uses *QualitativeLabel* values for the evaluation, and hence uses the qualitative contribution attribute of *Contribution*, the qualitative importance (importance) attribute of *IntentionalElement*s, and the attribute qualitativeVal of intentional elements initialized from the selected *EvaluationStrategy*. The qualitative contributions (see Figure 1d) are *Make*, *SomePositive*, *Help*, *Unknown*, *Hurt*, *SomeNegative* and *Break*. The qualitative evaluation labels (see Figure 1c) are *Satisfied*, *WeaklySatisfied*, *None*, *WeaklyDenied*, *Denied*, *Conflict* and *Undecided*. The qualitative importance values (see Figure 3) are *High*, *Medium*, *Low* and *None*. Since these values are discrete, the propagation algorithm must consider them individually. To this end, lookup tables and partial

orderings are often used to define necessary functions explicitly. Same as quantitative evaluation algorithm, first, algorithm calculates decomposition, then contribution and at the end dependency links.

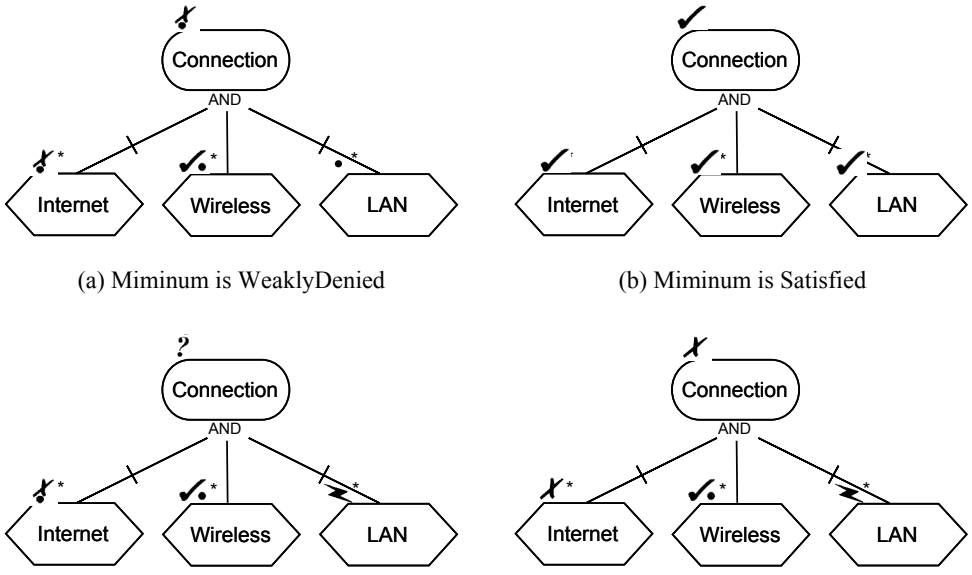
4.4.1 Calculating qualitative evaluations for decomposition links

This corresponds to the CalculateDecompositions(element) step in Figure 6. Similar to the quantitative algorithm, the result depends on the type of decomposition (AND, IOR, or XOR).

The satisfaction level of an intentional element with an AND-type decomposition link is the *minimum* value of the qualitative evaluation values of its source elements, where qualitative values are ordered from minimum to maximum as follows:

Denied < (*Conflict* = *Undecided*) < *WeaklyDenied* < *None* < *WeaklySatisfied* < *Satisfied*

However, *Conflict* results are substituted with *Undecided* as conflicts are not propagated. This simplifies the discovery of root causes (the first conflict) during the analysis of complex models. Figure 12 provides four examples of qualitative AND-type decomposition that illustrate this propagation.



(a) Miminum is WeaklyDenied (b) Miminum is Satisfied

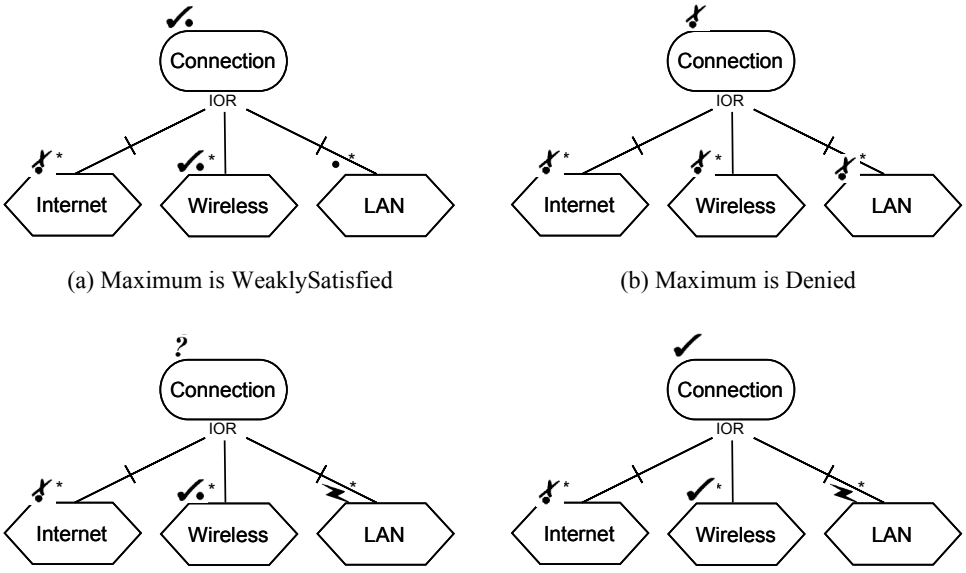
(c) Minimum is Conflict: Undecided is propagated (d) Miminum is Denied, even if Conflict is present

Figure 12 Example: Qualitative evaluation of AND-type decomposition links

For an IOR-type decomposition, the satisfaction level is the *maximum* value of the quantitative evaluation of its source elements, where qualitative values are ordered from minimum to maximum as follows:

Denied < *WeaklyDenied* < *None* < *WeaklySatisfied* < (*Conflict* = *Undecided*) < *Satisfied*

Again, *Conflict* results are substituted with *Undecided* as conflicts are not propagated. Figure 13 provides four examples of qualitative IOR-type decomposition that illustrate this propagation.



(a) Maximum is WeaklySatisfied (b) Maximum is Denied

(c) Maximum is Conflict: Undecided is propagated (d) Maximum is Satisfied, even if Conflict is present

Figure 13 Example: Qualitative evaluation of IOR-type decomposition links

For an XOR-type decomposition link, the *maximum* is propagated in the same way as for an IOR-type decomposition, but a warning is generated if more than one source element have a quantitative evaluation value different from *None*.

4.4.2 Calculating qualitative evaluations for contribution links

This corresponds to the CalculateContributions(element, decompValue) step in Figure 6. Correlations are treated the same way as contributions. However, unlike the quantitative evaluation of contribution links, there is no notion of tolerance here. The algorithm is presented in Figure 14.

Algorithm CalculateContributions **Inputs** element:IntentionalElement, decompValue:QualitativeLabel **Output** contribValue:QualitativeLabel oneCont:QualitativeLabel *// one weighted contribution* ns:Integer = 0 *// number of Satisfied weighted contributions* nws:Integer = 0 *// number of WeaklySatisfied weighted contributions* nwd:Integer = 0 *// number of WeaklyDenied weighted contributions*

```
nd:Integer = 0                      // number of Denied weighted contributions
nu:Integer = 0                      // number of Undecided weighted contributions
weightSD:QualitativeLabel           // partial weighted contribution from ns and nd
weightWSWD:QualitativeLabel         // partial weighted contribution from nws and nwd

// adjust the weighted contribution counters according to decompValue
AdjustContributionCounters(decompValue, ns, nws, nwd, nd, nu)

// compute the numbers of weighted contributions for each kind
for each link:Contribution in element.linksDest
{
        oneCont = WeightedContribution(link.src.qualitativeVal, link.contribution)
        AdjustContributionCounters(oneCont, ns, nws, nwd, nd, nu)
}

// check for the presence of undecided weighted contributions
if (nu > 0)
        contribValue = Undecided
else
{
        weightSD = CompareSatisfiedAndDenied (ns, nd)
        weightWSWD = CompareWSandWD (nws, nwd)
        contribValue = CombineContributions (weightSD, weightWSWD)
}

return contribValue
```

Figure 14 Example: Qualitative CalculateContributions algorithm

The AdjustContributionCounters algorithm (Figure 15) is first invoked by CalculateContributions to increment the weighted contribution counter that corresponds to decompValue. It is then invoked in the *for* loop to increment the counters for each individual weighted contribution computed from contribution links.

```
Algorithm AdjustContributionCounters
Inputs qualValue:QualitativeLabel
Modifies ns, nws, nwd, nd, nu:Integer

case qualValue of
        Satisfied:          ns++
        WeaklySatisfied:    nws++
        WeaklyDenied:       nwd++
        Denied:             nd++
        Undecided:          nu++
```

Figure 15 Example: AdjustContributionCounters algorithm

The CalculateContributions algorithm also uses a WeightedContribution function that computes one qualitative weighted contribution according to the lookup table in Table 2, where the rows specify the possible qualitative evaluation values of the source and where the

columns specify the possible qualitative contribution types of the element's incoming contribution link. Note that previously found conflicts are not propagated by this function, which propagates *Undecided* instead.

Table 2 WeightedContribution function for the computation of one weighted contribution

	Make	***Help***	***SomePositive***	***Unknown***	***SomeNegative***	***Hurt***	***Break***
Denied	Denied	WeaklyDenied	WeaklyDenied	None	WeaklySatisfied	WeaklySatisfied	Satisfied
WeaklyDenied	WeaklyDenied	WeaklyDenied	WeaklyDenied	None	WeaklySatisfied	WeaklySatisfied	WeaklySatisfied
WeaklySatisfied	WeaklySatisfied	WeaklySatisfied	WeaklySatisfied	None	WeaklyDenied	WeaklyDenied	WeaklyDenied
Satisfied	Satisfied	WeaklySatisfied	WeaklySatisfied	None	WeaklyDenied	WeaklyDenied	Denied
Conflict	Undecided	Undecided	Undecided	Undecided	Undecided	Undecided	Undecided
Undecided	Undecided	Undecided	Undecided	Undecided	Undecided	Undecided	Undecided
None	None	None	None	None	None	None	None

If there is at least one *Undecided* weighted contribution detected, then the result is *Undecided*. Otherwise, three functions will be used in sequence to compute the result.

The CompareSatisfiedAndDenied function determines if there are *Satisfied* values without *Denied* values, or the opposite. If none of these values are present, then *None* is returned. However, if there is at least one of each, then *Conflict* returned. Formally:

CompareSatisfiedAndDenied (ns, nd) = *Conflict*, **if** (ns > 0 **and** nd > 0)
= *Satisfied*, **if** (ns > 0 **and** nd = 0)
= *Denied*, **if** (nd > 0 **and** ns = 0)
= *None*, **if** (ns = 0 **and** nd = 0)

The CompareWSandWD function determines if there are more *WeaklySatisfied* values than *WeaklyDenied* values, or the opposite. If their numbers are equal, then these contributions cancel each other out and *None* is returned. Formally:

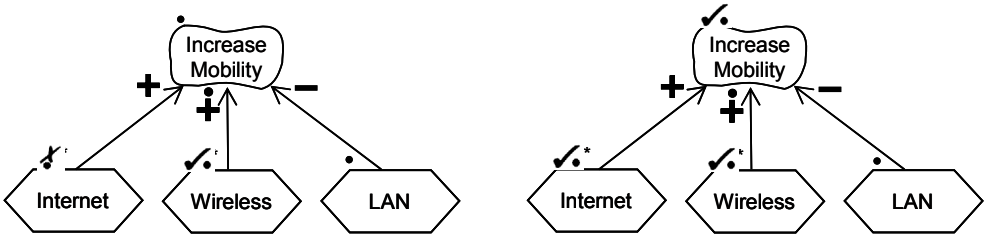
CompareWSandWD (ws, wd) = *WeaklySatisfied*, **if** (nws > nwd)
= *WeaklyDenied*, **if** (nwd > nws)
= *None*, **if** (nwd = nws)

The final result is computed with the CombineContributions function, which combines the previously computed values according to Table 3. In this table, the rows specify the possible qualitative values representing the global influence of weak contributions (i.e., weightWSWD), whereas the columns specify the possible qualitative values representing the global influence of *Satisfied* and *Denied* contributions (i.e., weightSD).

Table 3 CombineContributions function for the computation of the final contribution

	Denied	***Satisfied***	***Conflict***	***None***
Weakly Denied	Denied	Weakly Satisfied	Conflict	Weakly Denied
Weakly Satisfied	Weakly Denied	Satisfied	Conflict	Weakly Satisfied
None	Denied	Satisfied	Conflict	None

Figure 16 provides two examples with three contributions each (the initial decompValue is *None*). Strategies initialize two elements in each example. In (a), we have (*WeaklyDenied* × *SomePositive*) = *WeaklyDenied*, (*WeaklySatisfied* × *Make*) = *WeaklySatisfied*, and (*None* × *SomeNegative*) = *None*. The comparison of *Satisfied* and *Denied* results in a 0:0 tie and therefore *None*. The comparison of *WeaklySatisfied* and *WeaklyDenied* results in a 1:1 tie and therefore *None*. Finally, the combined contribution of *None* and *None* results in *None*. In (b), we have (*WeaklySatisfied* × *SomePositive*) = *WeaklySatisfied*, (*WeaklySatisfied* × *Make*) = *WeaklySatisfied*, and (*None* × *SomeNegative*) = *None*. The comparison of *Satisfied* and *Denied* results in a 0:0 tie and therefore *None*. The comparison of *WeaklySatisfied* and *WeaklyDenied* results in a 2:0 win and therefore *WeaklySatisfied*. Finally, the combined contribution of *None* and *WeaklySatisfied* results in *WeaklySatisfied*.



(a) Contributions: None is propagated (b) Contributions: WeaklySatisfied is propagated

Figure 16 Example: Qualitative evaluation of contribution links

4.4.3 Calculating qualitative evaluations for dependency links

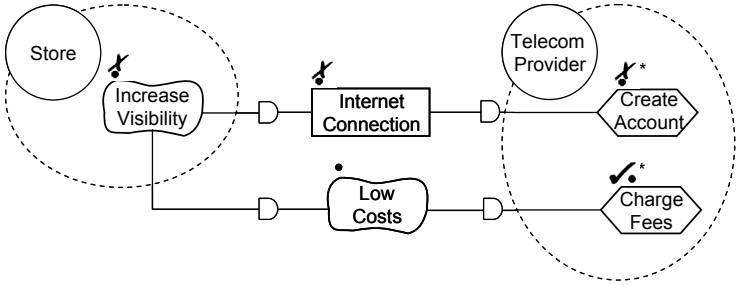
This corresponds to the CalculateDependencies(element, contribValue) step in Figure 6. In this algorithm, the source element of the dependency links cannot have an evaluation value higher than those of the intentional elements it depends on (i.e., the target elements of the dependency links). This algorithm hence simply returns the *minimum* value between contribValue and the qualitative evaluation values of the target elements. The qualitative

values are ordered from minimum to maximum in the same way as for qualitative AND-type decompositions:

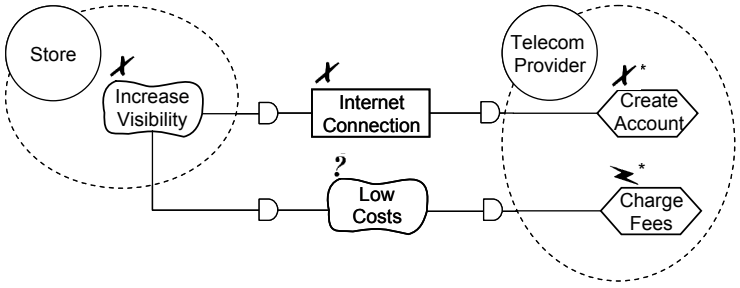
Denied < (*Conflict* = *Undecided*) < *WeaklyDenied* < *None* < *WeaklySatisfied* < *Satisfied*

Again, *Conflict* results are substituted with *Undecided* as conflicts are not propagated.

Two examples are shown in Figure 17, with strategies that initialize the two tasks. Consequently, the qualitative values of other elements are initially set to *None*. Example (a) is similar to the one from Figure 10. Internet Connection becomes *WeaklyDenied* since this value is less than *None*. Low Costs, on the other hand, will keep its value of *None* because it is less than *WeaklySatisfied*. The Increase Visibility softgoal gets the value *WeaklyDenied* because this is the minimum between *None*, *None*, and *WeaklyDenied*. Example (b) illustrates that a *Conflict* value in a target element propagates to an *Undecided* value in the source element (e.g., Low Cost), unless there is a *Denied* value in another target element or in contribValue (in which case the propagated value is *Denied*, e.g. for Increase Visibility).



(a) Minimum is WeaklyDenied



(b) Minimum is Denied, even if Conflict is present

Figure 17 Example: Qualitative evaluation of dependency links

4.4.4 Calculating qualitative evaluations for actors

This is the third and last step discussed in Section 4.2. In order to compute the qualitative evaluation value of an actor, the qualitative satisfaction value and qualitative importance value of each intentional element bound to the actor are used.

The CalculateActorEvaluation algorithm (Figure 18) is similar to the qualitative CalculateContributions algorithm (Figure 14) and reuses some of its sub-algorithms.

```
Algorithm CalculateActorEvaluation
Inputs actor:Actor
Output actorEvalValue:QualitativeLabel

oneElemVal:QualitativeLabel      // 1 elem. value weighted according to its importance
ns:Integer = 0                   // number of Satisfied weighted values
nws:Integer = 0                  // number of WeaklySatisfied weighted values
nwd:Integer = 0                  // number of WeaklyDenied weighted values
nd:Integer = 0                   // number of Denied weighted values
nu:Integer = 0                   // number of Undecided weighted values
nc:Integer = 0                   // number of Conflict weighted values
weightSD:QualitativeLabel        // partial weighted values from ns and nd
weightWSWD:QualitativeLabel      // partial weighted values from nws and nwd

// compute the numbers of weighted contributions for each kind
for each boundElem:IntentionalElement in actor.elems
{
        oneElemVal=WeightedImportance(boundElem.qualitativeVal,
                                      boundElem.importance)
        AdjustEvaluationCounters(oneElemVal, ns, nws, nwd, nd, nu, nc)
}

// check for the presence of undecided and conflict weighted evaluation values
if (nc > 0)
        actorEvalValue = Conflict
else if (nu > 0)
        actorEvalValue = Undecided
else
{
        weightSD = CompareSatisfiedAndDenied (ns, nd)
        weightWSWD = CompareWSandWD (nws, nwd)
        actorEvalValue = CombineContributions (weightSD, weightWSWD)
}

return actorEvalValue
```

Figure 18 Example: Qualitative CalculateActorEvaluation algorithm

First, the qualitative evaluation value of each intentional element bound to the actor is weighted according to the importance of that element to the actor. This WeightedImportance

function is defined in Table 4, where the rows specify the possible qualitative importance values of the element and where the columns specify the possible qualitative evaluation values of the element. The AdjustEvaluationCounters function is similar to the AdjustContribution-Counters function (see Figure 15) but also increments the *nc* counter if a *Conflict* is provided as a qualitative value input. Then, the qualitative evaluation value of the actor is calculated with same sub-algorithms used to combine the qualitative weighted evaluation values for contribution links (i.e., CompareSatisfiedAndDenied, CompareWSandWD, and CombineContributions).

Table 4 WeightedImportance function for the computation of one element value

	Denied	***WeaklyDenied***	***WeaklySatisfied***	***Satisfied***	***Conflict***	***Undecided***	***None***
High	Denied	WeaklyDenied	WeaklySatisfied	Satisfied	Conflict	Undecided	None
Medium	WeaklyDenied	WeaklyDenied	WeaklySatisfied	WeaklySatisfied	Conflict	Undecided	None
Low	WeaklyDenied	None	None	WeaklySatisfied	Conflict	Undecided	None
None	None	None	None	None	None	None	None

For example, Figure 19 shows an actor with four softgoals, three of which with importance other than *None*. The recalculated, qualitative evaluation values are:

- Reliability: WeightedImportance (*High*, *Satisfied*) = *Satisfied*
- Low Cost: WeightedImportance (*Low*, *Satisfied*) = *WeaklySatisfied*
- High Perf: WeightedImportance (*Medium*, *WeaklyDenied*) = *WeaklyDenied*
- Low Weight: WeightedImportance (*None*, *WeaklySatisfied*) = *None*

The comparison of *Satisfied* and *Denied* results in a 1:0 win and therefore *Satisfied*. The comparison of *WeaklySatisfied* and *WeaklyDenied* results in a 1:1 tie and therefore *None*. Finally, the combined contribution of *Satisfied* and *None* results in the actor evaluation of *Satisfied*.

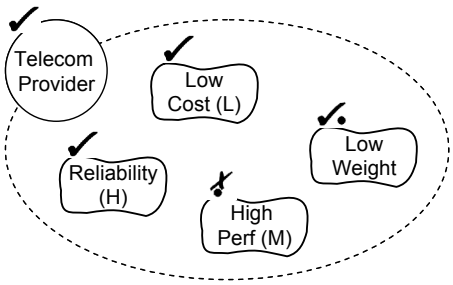


Figure 19 Example: Qualitative evaluation of actors

4.5 A Hybrid Evaluation Algorithm

This hybrid GRL algorithm uses *Integer* values for the evaluation, and hence uses the importanceQuantitative attribute of *Actors* and the new quantitativeVal attribute of intentional elements initialized from the selected *EvaluationStrategy*. However, unlike the quantitative evaluation algorithm seen in Section 4.3, the hybrid algorithm uses the qualitative contribution attribute of *Contribution*. A conversion table defines a mapping from qualitative contributions to integer values representing an equivalent quantitative contribution. Once this conversion is done, the rest of the algorithm is similar to the one in Section 4.3.

This is an example where quantitative and qualitative values are mixed. In this example, the discrete scale for contributions has 7 levels instead of 201 levels ([-100..100]) as in the quantitative evaluation algorithm. This may improve the usability of models in domains where the weight of contributions cannot easily be determined with precision.

4.5.1 Calculating hybrid evaluations for decomposition links

This corresponds to the CalculateDecompositions(element) step in Figure 6. The algorithm is the same as the quantitative algorithm in Section 4.3.1.

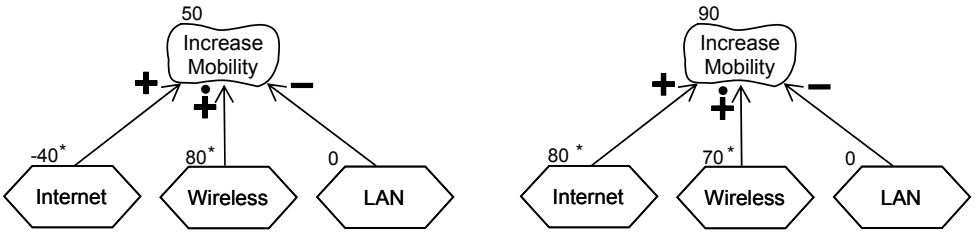
4.5.2 Calculating hybrid evaluations for contribution links

This corresponds to the CalculateContributions(element, decompValue) step in Figure 6. The algorithm first maps all qualitative contributions to quantitative contributions using Table 5. The content of this table reflects the relative ordering of qualitative contributions, however the associated quantitative numbers could be defined otherwise (e.g., 67 instead of 75, 33 instead of 25, etc.). When converting approximate qualitative labels to more precise measures, we run the risk of inserting a precision into our results which does not derive from an authoritative source, such as a concrete domain measure. It is important to keep in mind, when using hybrid methods, that the quantitative results are finer-grained approximations and not precise measures. Once all values are integers, the algorithm seen for the quantitative evaluation in Section 4.3.2 is used.

Table 5 Quantitative contribution values for qualitative contributions

Qualitative Contribution	*Quantitative Contribution*
Make	100
SomePositive	75
Help	25
Unknown	0
Hurt	-25
SomeNegative	-75
Break	-100

Figure 20 provides two examples with three contributions each (the initial decompValue is 0). Strategies initialize two elements in each example. The qualitative contributions are mapped to integer values according to Table 5. In (a), we have (-40 × 75) + (80 × 100) + (0 × -75) = 50. In (b), where the tolerance has been set to 10, we have (80 × 75) + (70 × 100) + (0 × -75) = 130. However, as there is no fully satisfied weighted contribution, then 100-tolerance = 90 is output.



(a) Contributions (b) Contributions with a tolerance of 10

Figure 20 Example: Hybrid evaluation of contribution links

4.5.3 Calculating hybrid evaluations for dependency links

This corresponds to the CalculateDependencies(element, contribValue) step in Figure 6. The algorithm is the same as the quantitative algorithm in Section 4.3.3.

4.5.4 Calculating hybrid evaluations for actors

This is the third and last step discussed in Section 4.2. The algorithm is the same as the quantitative algorithm in Section 4.3.4.

4.6 Comparison Between the Three Algorithm Types

In this section, we compare the three algorithms previously mentioned (quantitative, qualitative and hybrid) based on the usage context as well as on the criteria mentioned in Section 3.

A quantitative algorithm can be used in situations where quantitative information about contributions exists (which is usually the case for mature goal models) and where initial satisfaction levels are detailed enough to justify the use of a quantitative scale. When there is insufficient information about the domain (e.g. when the model is not mature enough) and when satisfaction levels can only be describe in terms of being positive/negative and sufficient/insufficient, then a qualitative algorithm is likely to be the best option (as a quantitative algorithm may distinguish between options that should not be distinguished given the high level of uncertainty in the model and the initial satisfaction levels). If only some of the information is available quantitatively, then a hybrid algorithm will likely be the best option. Note that as the goal model becomes more mature and detailed and as we progress through the development process, modellers can choose to move from a qualitative algorithm to a hybrid one and then to a quantitative one in order to get more precise results.

In Section 3, we defined 12 criteria for the evaluation algorithms: evaluation types, propagation direction, actor satisfaction, automations, cycles, conflicts, strategy consistency, evaluation overriding, relationships to UCM scenarios, evaluation ordering for links, link evaluation functions, and tolerance. The three evaluation algorithms we introduced here cover the three different types of evaluation we can use in GRL (quantitative, qualitative and hybrid). These three algorithms also support forward propagation characteristics and they include actor satisfaction. A qualitative algorithm propagates conflict with an unknown satisfaction value, which needs to be resolved later (e.g. through interactions or an enhanced strategy definition). Therefore, our qualitative algorithm may be interactive whereas our quantitative and hybrid algorithms are fully automated. These three algorithms accept GRL models with cyclic links (except for a few exceptional topologies), but none allows for inconsistent strategies. The evaluation of dependency links happens at the end of the evaluation process and the value of an intentional element can be overridden during the calculation of dependency links. Our quantitative and hybrid algorithms can influence the traversal of UCM scenarios (and vice-versa) but a qualitative algorithm cannot. In all of the algorithms mentioned here, decomposition links are first calculated followed by contribution links and then dependency links. Our quantitative and hybrid algorithms use a tolerance

factor while qualitative algorithm does not use any tolerance factor. Table 6 summarize the comparison of the three algorithms based on the 12 criteria of Section 3.

Table 6 Comparison between our three evaluation algorithms

Criteria	*Qualitative Algo.*	*Quantitative Algo.*	*Hybrid Algo.*
Evaluation Type	Qualitative	Quantitative	Hybrid
Propagation Direction	Forward	Forward	Forward
Actor Satisfaction	Yes	Yes	Yes
Automation	Interactive	Fully Automated	Fully Automated
Cycles	Partial Support	Partial Support	Partial Support
Conflicts	Detect Conflict	No Conflict	No Conflict
Strategy Consistency	Consistent	Consistent	Consistent
Evaluation Overriding	Yes	Yes	Yes
Relationship to UCM	No	Yes	Yes
Link Evaluation Ordering	Dec., Cont., Dep.	Dec., Cont., Dep.	Dec., Cont., Dep.
Link Evaluation Functions	See 4.4	See 4.3	See 4.5
Tolerance Factor	No	Yes	Yes

4.7 Implementation in jUCMNav

In his thesis, Roy implemented the support for editing GRL models and for analysing them in the jUCMNav tool.[18,19] He provided an extensible architecture enabling the support of multiple evaluation algorithms for GRL models, and defined a hybrid algorithm as an example. For two years, this was the only analysis algorithm in place, and unfortunately its results were counter-intuitive in many situations. Taking advantage of the extensible nature of jUCMNav, we have recently implemented the three algorithms discussed in this chapter, in addition to Roy's. Users can select the algorithm they want to use for any URN model. Moreover, the tool synchronizes the quantitative and qualitative values input by modellers. For instance, if a quantitative satisfaction level of 100 is provided for an intentional element in a strategy, then the corresponding qualitative satisfaction level is automatically set to Satisfied. In this way, modellers can easily switch between evaluation algorithms for a given GRL model.

5. Example

In this section, six strategies are defined and applied to the telecommunication example introduced in Section 2.2. Their evaluation will enable the illustration and comparison of the three evaluation algorithms defined in Section 4. Our objective is not to determine the "best" algorithm, as different algorithms suit different analysis needs based on the information available, but to contrast several aspects of sample algorithms that share commonalities.

The main alternatives to be evaluated in this system concern the location of the service data (two choices, represented as two alternative tasks in the model) and the location of the service logic (also two choices) in the wireless network. To simplify our analysis, we will assume that the Maximum Hardware Utilisation is fixed for all strategies (evaluation = 50, qualitativeEvaluation = WeaklySatisfied). In addition, when placing the data in a new service node (SN) is favoured over placing the data in an existing service control point (SCP), then we will also consider the impact of the availability of new service nodes from the vendor, as a dependency exists. Therefore, the six GRL strategies we will explore are:

1. *ServiceInSwitchDataInSCP*: The service logic is in the switch and the data in the SCP.
2. *ServiceInSwitchDataInSNready*: The service logic is in the switch and the data in the SN, and the vendor can supply new SN.
3. *ServiceInSwitchDataInSNnotReady*: The service logic is in the switch and the data in the SN, but the vendor cannot supply new SN.
4. *ServiceAndDataInSCP*: The service logic and the data are both in the SCP.
5. *ServiceInSCPDataInSNnotReady*: The service logic is in the SCP and the data in the SN, but the vendor cannot supply new SN.
6. *ServiceInSCPDataInSNready*: The service logic is in the SCP and the data in the SN, and the vendor can supply new SN.

With jUCMNav, strategies can be evaluated for any available algorithm. Satisfaction levels are reported next to the intentional elements and actors. In addition, jUCMNav uses a color scheme to highlight intentional elements that are satisfied (green), neutral (yellow), or denied (red), contributing to the understanding of which goals are satisfied or not at a glance. For example, Figure 21 illustrates the result of evaluating the ServiceInSwitchDataInSCP strategy with the quantitative algorithm of Section 4.3 (and hence quantitative values are reported) whereas Figure 22 reports on the results of evaluating the ServiceAndDataInSCP strategy with the qualitative algorithm of Section 4.4 (and hence qualitative values are reported).

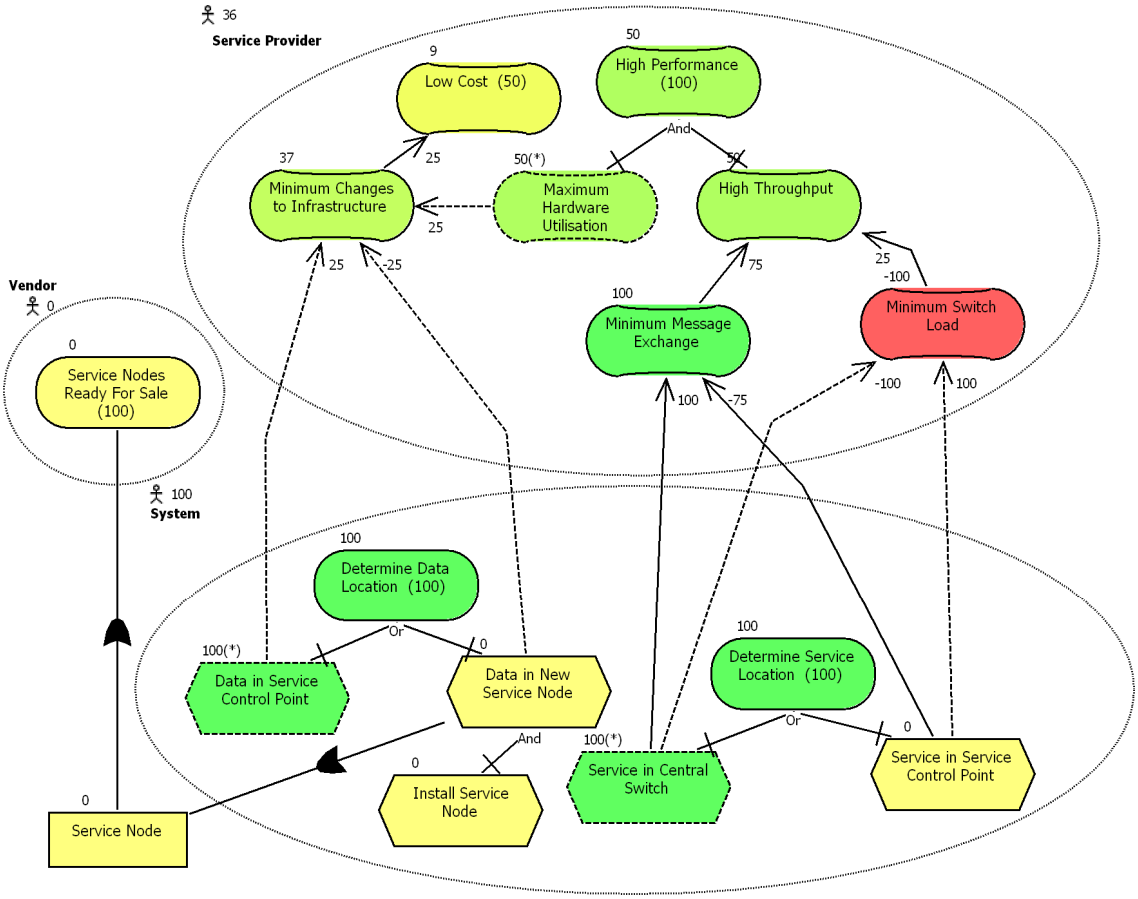


Figure 21 Quantitative evaluation of the ServiceInSwitchDataInSCP strategy

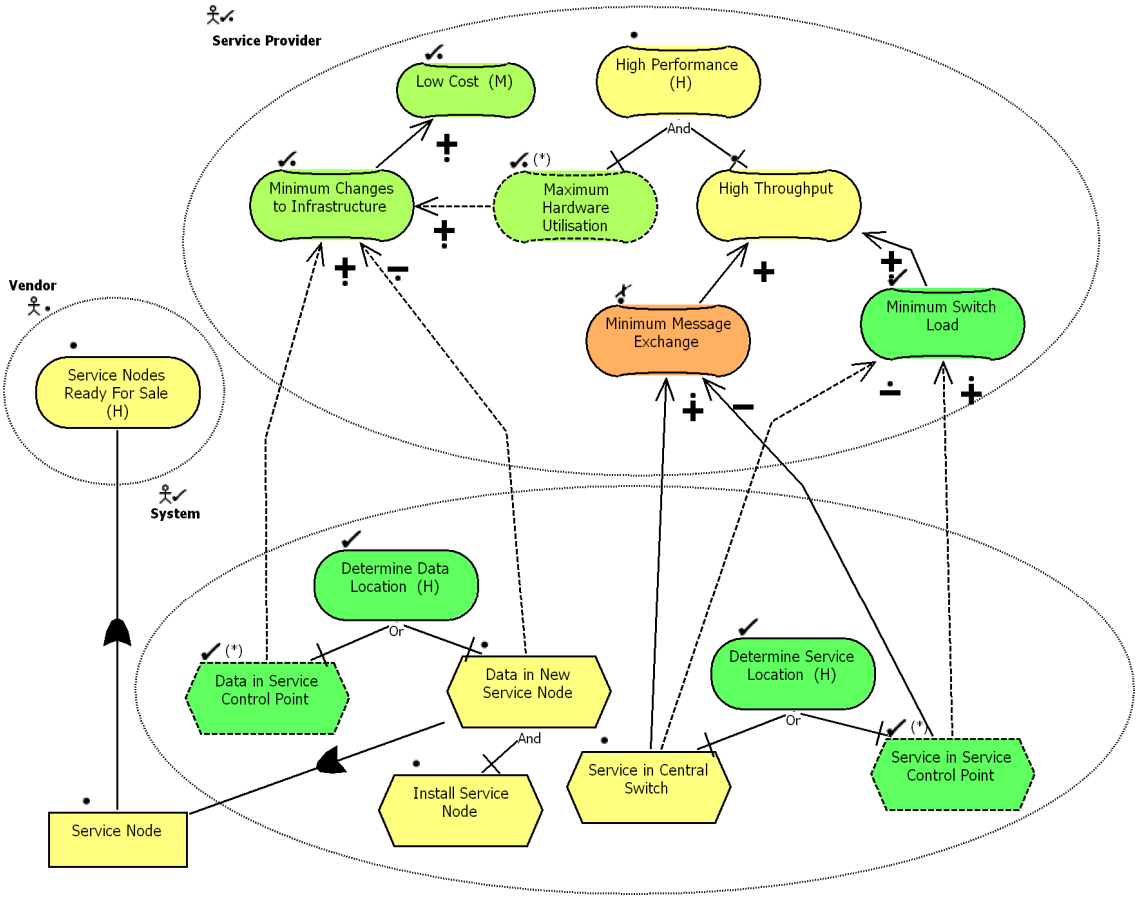


Figure 22 Qualitative evaluation of the ServiceAndDataInSCP strategy

The entire set of results for all six strategies and all algorithms is reported in Table 7. Qualitative and quantitative values prefixed with a star (*) are initialized by the corresponding strategy. Color coding is used to highlight what is satisfied and what is denied.

In this wireless system model, what we want to optimize is the satisfaction of the Service Provider actor, as long as the System's goals are fully met. The Vendor's satisfaction is of no concern to the Service Provider, who needs to make the appropriate decisions regarding the deployment of this new service on its network. The last three lines of Table 7 show the satisfaction results for all three actors. Although the results seem to generally agree across algorithms, several interesting differences can be observed:

Table 7 Evaluation results for six strategies and three algorithms

		Quantitative algorithm						Qualitative algorithm						Hybrid algorithm					
		1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
Intentional Elements	Low Cost (M)	9	-3	9	9	9	-3	WS	N	WS	WS	WS	N	10	-3	10	10	10	-3
	High Performance (H)	50	50	50	-31	-31	-31	N	N	N	N	N	N	50	50	50	-31	-31	-31
	Minimum Changes to Infrastructure	37	-13	37	37	37	-13	WS	N	WS	WS	WS	N	38	-12	38	38	38	-12
	Maximum Hardware Utilisation	*50	*50	*50	*50	*50	*50	*WS	*WS	*WS	*WS	*WS	*WS	*50	*50	*50	*50	*50	*50
	High Throughput	50	50	50	-31	-31	-31	N	N	N	N	N	N	50	50	50	-31	-31	-31
	Minimum Message Exchange	100	100	100	-75	-75	-75	S	S	S	WD	WD	WD	100	100	100	-75	-75	-75
	Minimum Switch Load	-100	-100	-100	100	100	100	D	D	D	S	S	S	-100	-100	-100	100	100	100
	Determine Data Location (H)	100	100	0	100	0	100	S	S	N	S	N	S	100	100	0	100	0	100
	Data in Service Control Point	*100	0	0	*100	0	0	*S	N	N	*S	N	N	*100	0	0	*100	0	0
	Data in New Service Node	0	100	-100	0	-100	100	N	S	D	N	D	S	0	100	-100	0	-100	100
	Service in Central Switch	*100	*100	*100	0	0	0	*S	*S	*S	N	N	N	*100	*100	*100	0	0	0
	Service in Service Control Point	0	0	0	*100	*100	*100	N	N	N	*S	*S	*S	0	0	0	*100	*100	*100
	Determine Service Location (H)	100	100	100	100	100	100	S	S	S	S	S	S	100	100	100	100	100	100
	Service Node	0	*100	-100	0	-100	*100	N	*S	D	N	D	*S	0	*100	-100	0	-100	*100
	Service Nodes Ready For Sale (H)	0	*100	*-100	0	*-100	*100	N	*S	*D	N	*D	*S	0	*100	*-100	0	*-100	*100
	Install Service Node	0	*100	*100	0	*100	*100	N	*S	*S	N	*S	*S	0	*100	*100	0	*100	*100
Actors	Service Provider	36	32	36	-17	-17	-21	WS	N	WS	WS	WS	N	36	32	36	-17	-17	-21
	System	100	100	50	100	50	100	S	S	S	S	S	S	100	100	50	100	50	100
	Vendor	0	100	-100	0	-100	100	N	S	D	N	D	S	0	100	-100	0	-100	100

Legend: *-initial value, D-Denied, WD-WeaklyDenied, N-None, WS-WeaklySatisfied, S-Satisfied

- Amongst the strategies evaluated, the quantitative and hybrid algorithms suggest that the first one (*ServiceInSwitchDataInSCP*) represents a set of decisions that is superior to the others (and would document the rationale for making these decisions requirements for the system). The qualitative algorithm does not disagree, but it would not be able to distinguish amongst strategies 1, 3, 4, 5 as the System is satisfied and the Service Provider weakly Satisfied in these cases.

- The results of the quantitative and hybrid algorithms for this specific model are almost exactly the same. This is because the model used quantitative contributions that match closely the corresponding values of their qualitative counterpart, as defined in Table 5. Had the quantitative contributions been more fine grained (e.g., by using 40 or 60 instead of 50 for some of the contributions), then the evaluation results would have been different.

- The qualitative model does not distinguish between slightly positive and slightly negative satisfactions for some of the high-level intentional elements (e.g., High

Performance is always evaluated to None) and actors (e.g., Service Provider for strategies 2 and 6) whereas the other two algorithms do.

- The sufficiency-oriented nature of the qualitative algorithm also leads to several disagreements with the other algorithms. For instance, the System actor is always fully satisfied (S) with the qualitative algorithm, because of the sufficient contribution of the other highly important goal, whereas it is only evaluated to 50 for strategies 3 and 5 in the other algorithms.

Of course, the precision of the analysis results highly depends on the precision of the values in the strategies and the contribution levels in the models. For some early decisions in the absence of precise knowledge about the system and its environment, qualitative values might be all we can afford. However, for a given algorithm and for similar assumptions, one can still provide an argument and a rationale for a strategy by comparing different strategies and by explaining the decision based on the comparative results.

6. Related Work

Over the past two decades, much effort was devoted to the development of goal-oriented languages, analysis techniques, and support tools. This section presents several key comparison points between our contributions and closely-related work only, and refers to other studies with a broader scope when available.

Some of the goal languages closest to GRL include KAOS, i*, NFR, and TROPOS. KAOS is a goal-oriented requirement engineering language and method used to capture requirements in terms of objects, goals, actions, constraints and agents.[3,4] Goals are either functional (services) or non-functional (quality of services). They can be assigned to agents, refined by other goals through AND/OR links, and operationalized by actions. Formal models and informal ones can be created. A quantitative algorithm can be used to evaluate the partial satisfaction of goals by computing the weighted average of the sub-goals' satisfaction. This is the quantitative version of the NFR qualitative algorithm. Although editors exist for KAOS models, including commercial tools (Objectiver[20] and its predecessor GRAIL[22]), analysis features do not yet support this evaluation algorithm. Matulevičius *et al.* provide an in-depth language-oriented comparison between KAOS and an earlier version of GRL.[21,23] Letier and van Lamsweerde also expand on the evaluation of KAOS models enriched with a probabilistic layer for reasoning about partial satisfaction.[14]

The *i** Framework[7] shares many concepts with GRL. However, it also contains many types of actors (e.g., agents, roles, and positions) and associations (e.g., generalization, instantiation, covers, occupies, is part of, and plays) that GRL does not differentiate. For the links that are defined in both languages, there are more restrictions on their usage in *i** than in GRL. *i** also distinguishes between strategic design models (where the focus is on actors and dependencies) and strategic rationale models (where the internal concerns of the actors are defined, hence providing a rationale for the dependencies). GRL, on the other hand, brings in strategies, a combination of qualitative and quantitative contributions, actor evaluations, generic URN links, and metadata. GRL also makes no distinction between strategic and rationale models, although both views can be expressed in different diagrams of a same GRL model. The OpenOME tool enables the creation and analysis of *i** models.[24] In the *i** framework, an interactive (semi-automated), forward propagation algorithm with qualitative values, defined by Horkoff *et al.*,[25] is currently supported. An interactive, backward propagation algorithm with qualitative values has been recently explored,[13] with plans to make it fully automated.

The NFR Framework also has concepts for intentional elements and qualitative contributions, as well as qualitative, forward propagation algorithm.[5] The qualitative algorithm is semi-automated and propagates conflicts when the evaluation values of the source nodes are not of the same type and are contradictory. OME is a tool that supports the NFR algorithm. However, it lacks the concepts of actors and dependencies found in *i** and GRL. Our GRL propagation algorithms handle these concepts, as well as quantitative evaluations.

TROPOS[8] is an agent-oriented language and software development methodology that includes the concepts of agents and goals. In the early requirements analysis phase, TROPOS adopts *i**'s modelling concepts and diagrams. There are four qualitative contribution levels in TROPOS (-, --, +, ++). TROPOS uses an axiomatization approach to formalize goal models. The language also supports both qualitative and quantitative relationships between goals, and can perform forward and backward propagation. These algorithms separate negative and positive evidence into two values during the propagation, whereas only one satisfaction value is propagated in our algorithms. Three types of conflicts (weak, medium, and strong) can be detected during qualitative evaluations, but they are left unresolved, i.e., negative and positive evidence are left separate. Unlike for the algorithms presented here (in

GRL), quantitative evaluations can also result in conflicts. Dependencies can be handled to a limited extent. Tool support for TROPOS exists for quantitative and qualitative analysis algorithms, but in the absence of the concept of strategy (à la GRL).

Ayala *et al.*[26] compare the abstract syntaxes of *i**, TROPOS, and a former version of GRL by highlighting noises, silences, ambiguities and contradictions. The standardized version of GRL in Recommendation Z.151 has addressed most of the issues discovered at the time. For additional comparative studies with a broader scope, several comparisons between goal-oriented tools are provided by Matulevičius *et al.*[27] and Roy.[18] For *i** tools and usage guidelines, the *i** Wiki[6] represents the best source of information. Grau *et al*. also review several *i**-based techniques.[28] A similar study was done by Anwer and Ikram.[29] From an algorithm perspective, Franch discusses approaches and challenges related to quantitative evaluation of goal/agent models in general,[12] and Horkoff's thesis compares many propagation procedures used to analyse goal models.[30]

The three recently implemented algorithms discussed in this paper address several issues of the original evaluation algorithm proposed by Roy.[18] Now, quantitative as well as qualitative evaluations are fully supported. The original evaluation algorithm (an hybrid one) normalized the initial evaluation labels which led to incorrect satisfaction values for the rest of the GRL model. This normalization step has been omitted for the new evaluation algorithms. Furthermore, Roy's algorithm reversed the order of some contribution types, thus violating the definitions in the new URN standard. This was also rectified. While the original evaluation mechanism differentiated between concepts of *priority* and *criticality*, the three new algorithms use the concept of *importance* only (as does the URN standard).

Overall GRL is rather unique in the sense that it is defined as a precise language that is open enough to support numerous types of rigorous analyses, including quantitative, qualitative, and hybrid ones. It is also the only language that supports actor evaluation and that integrates strategies as part of the language. The jUCMNav tool was also built from the start to be able to support multiple evaluation algorithms.

One other major advantage of GRL over other goal languages is its integration with the UCM scenario notation as part of the User Requirements Notation. In their pioneering work,[31] Liu and Yu explored the synergy between these two views in a requirements engineering process where goals help modellers find important scenarios and where scenarios help discover further goals. The benefits of combining GRL with UCM for business process

modelling were demonstrated by Weiss and Amyot.[32] Ghanavati further explored these concepts in the context of process compliance and evolution.[33] This work was extended for covering business process monitoring and performance management by Pourshahid *et al.*, with tool-supported extensions to GRL to support key performance indicators and online monitoring.[34] GRL strategies can also guide the selection of appropriate scenarios in the UCM view of a URN model; this integration at the analysis level was developed by Kealey[19] and was used to model context-aware and adaptive telecommunications services.[35]

The integration of GRL with other languages is not limited to UCM; Abid recently developed and prototyped a UML profile for GRL, enabling the integration of GRL concepts and diagrams with models defined with UML.[36]

Table 8 gives a brief summary of how popular goal-oriented modelling languages are supported by major analysis algorithm types and tools.

Table 8 Comparison between analysis approaches

	Bottom-Up Approach			***Top-Down Approach***		
	Qualitative Analysis	***Quantitative Analysis***	***Hybrid Analysis***	***Qualitative Analysis***	***Quantitative Analysis***	***Hybrid Analysis***
Modelling Language	TROPOS, GRL, KAOS	TROPOS, *i** Framework, NFR, GRL	GRL	TROPOS	TROPOS, *i** Framework	NA
Tool Support	GR-Tool, jUCMNav	GR-Tool, OME, OpenOME, jUCMNav	jUCMNav	GR-Tool, GOALSOLVE, GOALMINSOLVE	GR-Tool, OpenOME	NA

7. Conclusion

Goal models represent an essential tool for requirements engineers, system architects, and software developers, namely the Goal-oriented Requirement Language. However, their analysis and evaluation lead to many challenges when it comes time to develop and implement suitable algorithms for model evaluation. The major contribution of this paper is the provision of a general approach for addressing those challenges based on the standardization of the goal modelling language GRL as part of the ITU-T User Requirements Notation.

We provided an overview of the GRL concepts, metamodel, and graphical syntax, together with tool support provided by the jUCMNav Eclipse plug-in that can be used to support algorithms for model evaluation (Section 2).

We introduced criteria for characterizing analysis algorithms for goal model evaluation (Section 3) and introduced our approach for supporting such algorithms in GRL. It is worth noting that there exist many variation points that require design decisions when formalizing evaluation algorithms.

In Section 4, we formalized and illustrated three evaluation algorithms (one quantitative, one qualitative, and a hybrid one) that comply with the semantics of GRL and that satisfy minimal requirements for analysing goal models found in the URN standard.

We have implemented these new algorithms in the publicly available jUCMNav tool and provided ways for users to select the algorithm that they want to apply to a GRL model. This selection might depend on the maturity of the model and the quality of the information available to analyse it.

In Section 5, we illustrated the approach and the algorithms by evaluating six GRL strategies on a small telecommunication system example in order to highlight several similarities and differences amongst the three algorithms.

Section 6 discussed related work. It was observed that GRL embeds in the language concepts that are fairly unique and highly beneficial, such as strategies, actor evaluations, and an integration with UCM scenarios in URN.

This paper does not claim that the three algorithms presented here are "better" than others; they were primarily used to illustrate the complexity of the design of an evaluation mechanism for a goal-oriented language and to highlight particular features of the GRL language, including its flexibility.

This paper did not discuss the process of generating a good goal model, which is often tightly coupled with requirements elicitation and analysis phases. Although a large body of knowledge and many success stories exist (see the URN Virtual Library[16] and the *i** Wiki[6]), more research is required to define systematic processes that would help generate the (GRL) models used by our analysis tools. Finding "intuitive" evaluation algorithms and determining which algorithm would be best to use in a given context (e.g., automated or interactive conflict resolution) also deserves a lot more attention, and empirical and/or usability studies

may be necessary in order to provide valuable answers. A tool like jUCMNav, where adding a new evaluation algorithm is quite simple, could play an important role in such studies.

As for the evolution of GRL itself, we foresee several possibilities, but mainly in two directions: better support for business process modelling and analysis,[34] and aspect-oriented extensions for GRL (and for URN in general), where we already have interesting preliminary results.[37] With respect to business process modelling, it would be useful to explore further the use of GRL on its own with other notations such as UML or the Business Process Modelling Notation (BPMN).

Acknowledgments

This research was supported by NSERC through their Discovery, Postgraduate, and Collaborative Health Research Projects (in collaboration with CIHR) programs.

References

1. Mylopoulos J, Chung L,Yu E. From object-oriented to goal-oriented requirements analysis. Commun. ACM 1999; 42(11):31–37.
2. van Lamsweerde A. Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. In: Proc. 12th IEEE Int. Requirements Engineering Conference (RE'04), IEEE CS; 2004. pp 4–7.
3. van Lamsweerde, A. Requirements Engineering: From Craft to Discipline. In: Proc. 16th ACM SigSoft Int. Symp. on the Foundations of Software Engineering (FSE'2008), Atlanta, USA; 2008.
4. van Lamsweerde A. Requirements engineering: From System Goals to UML Models to Software Specifications. John Wiley & Sons; 2009. 712 p.
5. Chung L, Nixon BA, Yu E, Mylopoulos J. Non-Functional Requirements in Software Engineering. Dordrecht: Kluwer Academic Publishers; 2000. 412 p.
6. i* Wiki [Online]. Available at http://istar.rwth-aachen.de/tiki-view_articles.php. Accessed on November 15, 2008.
7. Yu E. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proc. 3rd IEEE Int. Symp. on Requirements Engineering, Washington, USA: IEEE CS; 1997. pp 226–235.
8. Giorgini P, Mylopoulos J, Sebastini R. Goal-oriented requirements analysis and reasoning in the Tropos methodology. Eng. Appl. Artif. Intell. 2005; 18:159–171.
9. Amyot D. Introduction to the User Requirements Notation: Learning by Example. Comput. Networks 2003; 42(3):285–301.
10. ITU-T. Recommendation Z.150 (02/03): User Requirements Notation (URN) – Language requirements and framework. Geneva, Switzerland; 2003. 28 pages.
11. ITU-T. Recommendation Z.151 (09/08): User Requirements Notation (URN) – Language definition. Geneva, Switzerland; 2008. 206 pages.
12. Franch X. On the Quantitative Analysis of Agent-Oriented Models. In: Proc. 18th Int. Conf. on Advanced Information Systems Engineering (CAiSE'06): LNCS 4001; 2006. pp 495–509.

13. Horkoff J, Yu E. Qualitative, Interactive, Backward Analysis of i* Models. 3rd Int. i* Workshop, Recife, Brazil; 2008. pp 43–46.
14. Letier E, van Lamsweerde A. Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. In: Proc. 12th ACM Int. Symp. on the Foundations of Software Engineering (FSE'04), Newport Beach, USA; 2004. pp 53–62.
15. jUCMNav [Online]. Available at http://jucmnav.softwareengineering.ca/jucmnav/ Accessed on November 4, 2009.
16. URN Virtual Library [Online]. Available at http://www.UseCaseMaps.org/pub/ Accessed on November 15, 2008.
17. ITU-T. Recommendation Z.111 (09/08): Notations to Define ITU-T Languages. Geneva, Switzerland; 2008. 17 pages.
18. Roy JF. Requirement Engineering with URN: Integrating Goals and Scenarios. M.Sc. thesis, SITE, University of Ottawa, Canada; March 2007.
19. Roy JF, Kealey J, Amyot D. Towards Integrated Tool Support for the User Requirements Notation. In: SAM 2006: Language Profiles – Fifth Workshop on System Analysis and Modelling, Kaiserslautern, Germany: LNCS 4320; 2006. pp 198–215.
20. Objectiver [Online]. Available at http://www.objectiver.com. Accessed on November 15, 2008.
21. Darimont R, Delor E, Massonet P, van Lamsweerde A. GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering. In: Proc. 19th Int. Conf. on Software Engineering, Boston, USA; 1997. pp 612–613.
22. Ballant D, Belpaire C, Darimont R, Delor E, Genard D, Nève C, Roussel JL, Vanbrabant A. Requirements Engineering with GRAIL/KAOS: From Goal Analysis to Automatically Derived Requirements Documents. In: Proc. 11th IEEE Int. Requirements Engineering Conf.; 2003.
23. Matulevičius R, Heymans P, Opdahl AL. Comparing GRL and KAOS using the UEML Approach. In: Enterprise Interoperability II: New Challenges and Approaches: Springer; 2007. pp 77–88.
24. OpenOME [Online]. Available at https://se.cs.toronto.edu/trac/ome/ Accessed on November 15, 2008.
25. Horkoff J, Yu E, Liu L. Analyzing Trust in Technology Strategies. In: Int. Conf. on Privacy, Security and Trust (PST 2006), Markham, Canada; 2006.
26. Ayala CP, Cares C, Carvallo JP, Grau G, Haya M, Salazar G, Franch X, Mayol E, Quer C. A Comparative Analysis of i*-based Agent-oriented Modelling Languages. In: Int. Workshop on Agent-Oriented Software Development Methodologies (AOSDM '05), Taipei, China; 2005. pp 43–50.
27. Matulevičius R, Heymans P, Sindre G. Comparing Goal-modelling Tools with the RE-tool Evaluation Approach. Inform. Tech. and Control, Kaunas, Technologija 2006; 35A(3):276–284.
28. Grau G, Cares C, Franch X, Navarrete F. A Comparative Analysis of i* Agent-Oriented Modelling Techniques. In: 18th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'06), San Francisco, USA; 2006. pp 657–663.
29. Anwer S, Ikram N. Goal Oriented Requirement Engineering: A Critical Study of Techniques. In: Proc. 13th Asia Pacific Software Engineering Conference (APSEC 2006); 2006. pp 121–130.
30. Horkoff J. Using i* Models for Evaluation. M.Sc. thesis, Dept. of Computer Science, University of Toronto, Canada; May 2006.
31. Liu L, Yu E. Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. Inform. Syst. 2004; 29(2):187–203.
32. Weiss M, Amyot D. Business Process Modeling with URN. Int. J. E-Business Res. 2006; 1(3):63–90.
33. Ghanavati S. A Compliance Framework for Business Processes Based on URN. M.Sc.

thesis, SITE, University of Ottawa, Canada; May 2007.
34. Pourshahid A, Chen P, Amyot, D., Forster, A.J., Ghanavati, S., Peyton, L., and Weiss, M., Business Process Management with the User Requirements Notation. Electron. Commer. Res. 2009; 9(4). To appear.
35. Amyot D, Becha H, Bræk R, Rossebø JEY. Next Generation Service Engineering. In: ITU-T Innovations in NGN Kaleidoscope Conf., Geneva, Switzerland: IEEE CS; 2008. pp 195–202.
36. Abid MR, Amyot D, Somé SS, Mussbacher G. A UML Profile for Goal-Oriented Modeling. In: SDL 2009: Design for Motes and Mobiles, 14th SDL Forum Conf., Bochum, Germany: LNCS 5719; 2009. pp 133–148.
37. Mussbacher G, Amyot D. Extending the User Requirements Notation with Aspect-oriented Concepts. In: SDL 2009: Design for Motes and Mobiles, 14th SDL Forum Conf., Bochum, Germany: LNCS 5719; 2009. pp 305–316.