Configurable Satisfiability Propagation for Goal Models Using Dynamic Compilation Techniques[1]

Elena Navarro[*] Patricio Letelier[**], David Reolid[*] and Isidro Ramos[**]

[*]Computing Systems Department, UCLM, EPSA, Campus Universitario s/n, Albacete, 02071, Spain, [enavarro|dreolid] @info-ab.uclm.es
[**]Department of Information Systems and Computation, UPV, Camino de Vera s/n, 46022, Valencia, Spain, [letelier|iramos]@dsic.upv.es

1 Introduction

It is frequently the case that at early stages of the requirements engineering process, critical decisions about what the system should provide are taken. Stakeholders and developers must evaluate alternatives and conflicts among the system requirements. In addition, a great deal of work must be done through focused brainstorming, validation, negotiation, and decision-making associated to vague or not completely defined requirements. In this context, Goal-Oriented modeling techniques emerge as a suitable way of defining and analyzing requirements, but also as an effective way to provide the necessary traceability towards other derived software artifacts.

This work aims at introducing a framework for exploiting Goal Models that allows the analyst to customize the analysis mechanisms according to the project needs. Our approach is based on the propagation algorithm proposed by (Giorgini et al. 2003), which establishes the essential computation of propagation. Using metamodeling techniques we provide the analyst with extensibility and customization mechanisms to modify the computation according to particular Goal Model elements, application domain, business rules, etc. These facilities are supplied by MORPHEUS, a tool we have developed for supporting our proposal.

[1] This work has been funded by the Spanish CICYT project DYNAMICA TIC2003-07776-C02-02.

The remainder of this work is structured as follows. In section 2 a brief introduction to Goal Models and their analysis capabilities for requirements is presented. Section 3 describes our proposal presenting the integration of metamodel elements in rule description that are used in a propagation algorithm. Section 4 describes MORPHEUS, a tool developed to give support to our framework and especially an add-in incorporated to exploit the Goal Model by means of satisfaction/denegation propagation. Eventually, in section 5 conclusions and future work are presented.

2 Background

A Goal Model is built as a directed graph by means of a refinement from the systems *goals* (or concerns). This refinement lasts until goals have enough granularity and detail so as to describe an *operationalization*, i.e., a solution that provides the target system to meet users' needs and expectations. This refinement process is performed by using *AND/OR* refinement relationships. An AND (OR) relationship between a goal $Goal_X$ and a set of sub-goals $G_1 \ldots G_N$ is established if the whole set of (at least one) sub-goals has to be satisfied in order to satisfy $Goal_X$. In addition, operationalizations are associated to the requirements (leaf goals) by means of *contribution* relationships that denote how they collaborate to achieve a goal.

Once a Goal Model is defined mechanisms can be used to analyse its satisfiability. The satisfaction (denegation) of a goal means that it will (will not) be provided by the system-to-be, i.e., user's needs and expectations will (will not) be met. The propagation to carry out this reasoning about goal satisfaction is addressed by means of two approaches:

- *Qualitative approach*. The idea is to establish positive or negative influence (for instance, by means of ++, +, #, -, -- symbols) of contributions from operationalizations to goals in the Goal Model. These operationalizations can be designs, agents, events on the market, etc., depending on the specific Goal Model that is being used. In this sense, the degree of satisfaction does not have a precise interpretation, i.e., it is not based on domain or system properties but on the analyst criteria. (Chung et al. 2000) and (Giorgini et al. 2003) are examples of this approach.
- *Quantitative approach*. In this case, weights are set to contribution relationships describing the satisfaction degrees that goals have among them. The propagation is performed in a similar way to the previous case, but now a specific value of satisfiability is achieved. Those weights can be assigned according to quite different criteria:
  1. *Subjective assignment* where only the analyst criteria is used to decide, as for instance (Giorgini et al. 2003)'s proposal.
  2. *Objective assignment*, which is based on domain properties. Some proposals are that presented by (Letier and Lamsweerde 2004) for reasoning about par-

tial satisfiability of requirement; or (Hansen et al. 1998) to analyze safety constraints of system-to-be with fault-trees.

Figure 1 shows a summary example of a Goal Model which has been defined within the European project Environmental Friendly and cost-effective Technology for Coating Removal (EFTCOR, 2003). Its aim is to design a family of robots capable of performing maintenance operations for ship hulls. On the left of the image, it can be observed how goals are refined from a high level goal (suitability) towards requirements (operationalizable goals) and operationalizations. For instance, it can be observed that to achieve a suitable system-to-be both "*ApproachRobot*" and "*CatchObject*" have to be satisfied; or how "*MoveUsingMUC*" positively contributes to "*MoveStepArms*". In addition, on the right side of the image the result of the propagation for that selected operationalizations is shown. The explanation of the used rules is presented in the next section.

Currently, there is no standard notation for goal-oriented specification but several proposals have appeared that address different activities and perspectives in the Requirements Engineering Process[2]. In this sense, Goal Models are mainly exploited for evaluating alternative designs such as those described by (Chung et al. 2000) (Letier and Lamsweerde 2004), business goals (Giorgini et al. 2003), etc. However, there is no consensus on which the most appropriate mechanism or proposal should be. For this reason, it is the analyst who finally has to make the final decision about which should be used for a specific project.

Similarly, the propagation of satisfaction/denegation through the Goal Model depends on the application domain and the expressiveness of the Goal Model (provided by its elements such as kinds of refinements, artefacts and associated attributes). Furthermore, the propagation rules could also be specific for the project or could even be modified in the same project in order to reflect some additional consideration during the analysis. This required flexibility is missing in the current

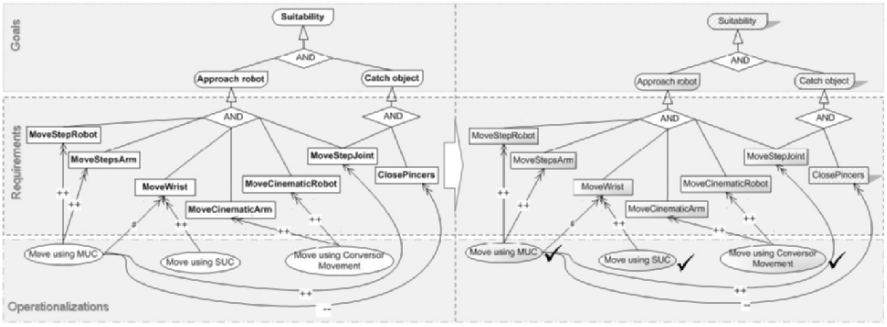

Fig. 1. Propagation of Satisfiability/Denegability(Selected Operationalization ✓, Satisfied Goal or Requirement Satisfied, Denied Goal or Requirement Denied)

[2] (Kavakli and Loucopoulos 2004) offer a detailed comparative framework about these proposals and their role in Requirements Engineering

proposals and constitutes an important obstacle while applying a specific Goal Model.

3 Our proposal

Our proposal took shape in the context of the ATRIUM methodology, presented by (Navarro et al. 2003), that guides the analyst through an iterative process, from a set of user/system needs to the instantiation of a Software Architecture. This proposal employs a Goal Model which is based on the (Dardenne et al. 1993) and (Chung et al. 2000) proposals. This Goal Model was extended by (Navarro et al. 2004) integrating the Aspect-Oriented approach, in order to achieve both the efficient management of crosscutting and the correct organization of the SRS (Software Requirement Specification). A set of techniques have been also incorporated to identify and specify *variability*, from the requirements stage, so that product lines and dynamic architectures can be dealt with. In this context, mechanisms for exploiting the ATRIUM Goal Model had to be defined and developed that do not only deal with specific expressiveness of the model but customize these techniques according to other specific needs.

In order to facilitate this customization a proposal for requirements metamodeling was developed and introduced in (Navarro et al. 2006). In this work, the most widely known notation for requirements specification (use cases, goal models, etc) were studied so as to identify the essential terms and concepts of each of them. Taking into account this study a metamodel for the essential concepts was defined that allows one to deal with generic expressiveness. The core concepts and their relationships are shown in Fig. 2. The first topic to consider was to define the metamodel is the description of artefacts. It allows one to describe any element to be included in the SRS. In this sense, any needed artefact can be described by inheriting from *Artefact* metaclass.

In addition to the artefact concept, it is also necessary to establish the artefact relationship with other artefacts of the SRS. Therefore, two kinds of relationships were identified. An artefact can be refined through other artefacts, forming a hierarchical structure. The basic kind of refinement included is *Refinement* that allows the analyst to define hierarchies of alternative specializations from the same parent and to relate one child to more than one parent by multiple inheritance. In addition

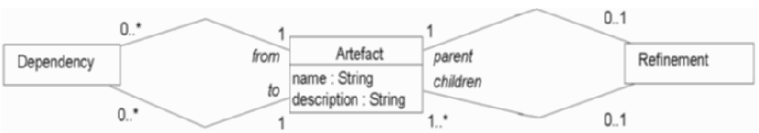

Fig. 2. Core Metamodel for Requirements Engineering

the *dependency* relationship has also been included in the metamodel. Perhaps this is the most conflictive relationship for consensus. For this reason, it is represented in its most generic form, i.e., by means of *Dependency* metaclass which is applicable to artefacts in the core. This metamodel has to be tailored according to the specific needs of expressiveness. With this purpose in mind, several steps were suggested to adapt and/or extend the metamodel. They need not be applied sequentially but in accordance with the analyst's preferences to describe new kinds of artefacts, relationships, attributes, constraints, etc, extending that described in the core.

In order to allow the analyst to customize the rules to be used during the propagation process an extension to the algorithm proposed by (Giorgini et al. 2003) has been developed. In this sense, the customization allows the analyst to include any kind of relationship and artefact along with their attributes to describe the propagation rules.

	$(G_2,G_3)\xrightarrow{and}G_1$	$G_2\xrightarrow{+S}G_1$	$G_2\xrightarrow{-S}G_1$	$G_2\xrightarrow{++S}G_1$	$G_2\xrightarrow{--S}G_1$
sat(G_1)	$\min\begin{Bmatrix} sat(G_2), \\ sat(G_3) \end{Bmatrix}$	$\min\begin{Bmatrix} sat(G_2), \\ P \end{Bmatrix}$	N	sat(G_2)	N
den(G_1)	$\min\begin{Bmatrix} den(G_2), \\ den(G_3) \end{Bmatrix}$	N	$\min\begin{Bmatrix} sat(G_2), \\ P \end{Bmatrix}$	N	sat(G_2)

Fig. 3. Qualitative Propagation rules described by (Giorgini et al. 2003), where $\xrightarrow{++S}$, $\xrightarrow{--S}$, etc. are describing contribution relationships

(Giorgini et al. 2003) have described a set of rules to specify how the propagation has to be compute that Fig. 3 depicts. Sat(G_i) and Den(G_i) specify the satisfiability and deniability, respectively, of the goal G_i, whose value is taken from the ordered set {"++","--","+", "-", "#"}. For example, let G_2 and G_3 subgoals of G_1 refined by using an AND relationship, the satisfiability of G_1 is set to the minimum of satisfiability of its subgoals. This means that G_1 is undefined, partially satisfied or totally satisfied depending on which minimum value between G_2 and G_3 is. As can be observed in Fig. 1 "*ApproachRobot*" is fully satisfied because each one of its subgoals is satisfied. Regarding contribution relationships, Fig. 3 depicts that an asymmetric propagation is performed. For instance, $G_2\xrightarrow{+S}G_1$ means that if G_2 is satisfied, then there is some evidence that G_1 is satisfied, but if G_2 is denied, then nothing is said about the satisfaction of G_1. On the contrary, if the relationship is $G_2\xrightarrow{-S}G_1$ then there is some evidence that G_1 is denied whether G_2 is satisfied, but if G_2 is satisfied, then nothing is said about the satisfaction of G_1. But Giorgini et al.'s have also described the rules for symmetric propagation. In that case, these rules consider the propagation of both satisfiability and deniability. An example could be the relationship $G_2\xrightarrow{+}G_1$ that means that if G2 is satisfied (denied), then there is some evidence that G1 is satisfied (denied).

Table 1. Propagation algorithm based on (Giorgini et al. 2003)'s proposal

```
label array Label_Graph(graph (G, R); label array Initial)
   Current=Initial;
   do
      Old=Current;
      for each Gi ∈ G do Current[i] = Update_Label(i, (G, R) Old);
   until (Current==Old);
   return Current;

label Update_Label(int i; graph (G, R); label array Old)
   for each Rj ∈ R s.t. target(Rj) == Gi do
      satij = Apply Rules Sat(Gi;Rj ;Old);                    //dropped
      denij = Apply Rules Den(Gi;Rj ;Old);                    //dropped
   return (maxj(maxj(satj);Old[i].sat), max(maxj(denij);Old[i].den))
      for each Ak∈ A AND Ak∈ set_valuable_attributes(Gi)       //added
         if applicable (Ak, Gi, Rj, Old)                      //added
               valuate (Ak, Gi, Rj, Old)                      //added
    return set_valuable_attributes(Gi);                       //added

boolean applicable(Attribute Ak;Artifact Gi;Relation Rj;      //added
labelarray Old)
   // depending on the specific rule it will check if         //added
   // the valuation can be performed                          //added

valuate(Attribute Ak; Artifact Gi; Relation Rj;               //added
        label array Old)                                      //added
   // depending on the specific rule it will perform          //added
   // the valuation of the attribute Ak                       //added
```

Table 1 shows how Giorgini et al.'s algorithm has been modified for customization purposes, highlighting the added pseudo-code with shadowed text and the dropped pseudo-code with a grey text. Two main functions were initially described by Giorgini et al.'s: "*Label_Graph*" which iterates through the goal model until there is no changes in the satisfibiality and deniability; and "*Update_Label*" which apply the appropriate rule, from those described in Fig. 3 and depending on the kind of relationship, to compute the satisfibiality and deniability of each node of the graph. As can be noticed, the initial proposal only describes the valuation for two attributes (sat and den) with a fixed set of rules (Fig. 3). However, with our proposal this set and attributes can be customized according to the specific needs of the project, as will be described below.

It is shown in Table 1 that whenever a rule has to be applied two steps must be performed. First, *applicable* describes which state or situation has to be satisfied in order to apply a specific rule. Second, *valuate* specifies the propagation computation when a rule is applied. In view of this, the grammar for defining both quantitative and qualitative rules is introduced by using the Backus-Naur Form (BNF). Table 2 and Table 3 show how applicable and valuate can be described, respectively. By using this grammar the rules defined by (Giorgini et al. 2003) can be easily described as shown in formulae (1) and (3).

Table 2. BNF for describing condition grammar

```
<condition> ::= '('<condition>')'<relational op>'('<condition>')'
                  | not '('<condition> ')' | <node> <logic op> <node>
<node>   ::= <identifier>|<expression>| <function>'('<identifier> ')'
<identifier>   ::= <id kind artefact>'.'<id attribute>
         | <id_ kind_relation>'.'<id_attribute>
<function>  ::= max | min | count | avg
<relational op>::= '&&' | '||' | '%%'
<logic op>::= '>' | '>=' | '=' | '!=' | '<=' | '<' | '>'
<expression>   ::= ''<string>'' | <number>
```

Table 3. BNF for describing valuation grammar

```
<valuation>::=id kind artefact>'.'<id kind enum attr>'=' <val enum>
      | <id kind artefact>'.'<id  kind numb attr> '=' <val  numb>
<val enum>  ::= <id kind artefact>'.'<id  kind enum attribute>
   |<expression_enum>
   |<funct_enum>'('<id_kind_artefact>'.'<id_ kind_enum_attribute> ')'
<funct enum>   ::= max | min
<val  numb>::='('<val numb>')'<op numb>'('<val numb>')'| <ident num>
               | <funct num> '(' <ident num> ')'| <expression num>
<funct_num> ::=  max | min | count | avg | sum | prod
<ident_num> ::= <id_kind_numbered_artefact>'.'<id_attribute>
               | <id_ kind_relation>'.'<id_attribute>
<op_numb>   ::= '+' | '-' | '*' | '/'
```

For instance, considering how the *CONTRIBUTION* relationship ($G_2 \xrightarrow{label} G_1$) is evaluated by Giorgini et al. we can appreciate that both the state of this relationship and the Goal source (G_S) are used to determine if the rule can be applied or not. In this sense, the condition could be described as: G_S*(satisfied) && label=--S*, i.e., G_S has an attribute that describes if G_S is satisfied. It is similarly applied to *label*, i.e., *CONTRIBUTION* relationship needs an attribute for specifying *--S* as its current state. In these terms, the best alternative is to represent these attributes following a syntax as described in Table 2 for <*identifier*>, i.e., by prefixing the attribute name with the name of the artefact or the relationship (see (1)).

In addition, when refinement relationships are considered, for instance an AND relationship ($(G_1 \mathrm{K}\, G_n) \xrightarrow{and} G_D$), some functions may be needed to determine the condition being applied to the set of artefacts G_1 to G_n. For this reason, an easy alternative is to use group functions as <*function*> describes in Table 2.

Furthermore, it is worthy of note that conditions can be combined to express other more complex ones. This will be described by using relational operations. This means that it would be possible to describe conditions such as:

$$(\text{Goal}_S.\text{satisfied} = \text{'Full'}) \text{ and } (\text{CONTRIBUTION.label}) = \text{'-S'} \quad (1)$$

$$(\text{Goal}_i.\text{type} = \text{'Peformance'}) \text{ and } (\text{Goal}_i.\text{priority} = \text{'High'}) \text{ and } \min(\text{Goal}_i.\text{satisfied} = \text{'Full'}) = 2 \quad (2)$$

In a similar maner, the syntax for the valuation is described by using the BNF (Table 3). As was stated in section 2, both quantitative and qualitative valuation should be described. For this reason, we have to distinguish between the two in order to make the peculiarities inherent in both kinds of valuations available. While describing qualitative valuations only enumerated attributes are made available to the analyst. This restriction is straightforward so that possible valuations are always constrained to a set of values. For instance, when considering the satisfiability, as described by Giorgini et al., the set *{Full, Partial, None}* is used. It also facilitates the valuation of this kind of attributes by describing functions for enumerations (<*function_enum*>). This requires that the set be defined as an ordered set in order to be able to properly apply these functions (min and max). This means that the valuation for an AND relationship (Fig. 3) could be easily described as appears in (3).

$$\text{Goal}_D.\text{satisfied} = \min(\text{Goal}_i.\text{satisfied}) \quad (3)$$

Related to the artefacts involved in a refinement relationship, aggregated functions (<*function_num*>) can be used for its treatment as described in:

$$\text{Goal}_D.\text{satisfied} = \text{sum}(\text{Goal}_i.\text{satisfied}) - \text{prod}(\text{Goal}_i.\text{satisfied}) \quad (4)$$

Some specific rules had to be introduced since ATRIUM Goal Model was defined to facilitate the analysis of variability. For instance, it has to be considered that whenever a *variation point* is described, its *multiplicity* must be defined, i.e., how many variants must exist at the same time in a product or architecture when the variability is being removed. Table 4 shows how rule OR relationship has been modified for dealing with variability.

Table 4. Defining rules for variability analysis

	Relationship	**Condition**	**Valuation**
Giorgini et al.	Satisfiability ($G_1 \ldots G_n \xrightarrow{OR} G_D$)		$\text{Goal}_D.\text{Sat} = \max(\text{Goal}_i.\text{Sat})$
Variability	Satisfiability ($G_1 \ldots G_n \xrightarrow{OR} G_D$)	(count(Goal_i.Sat==“+S”) + count(Goal_i.Sat==“++S”)) >= OR.multiplicity.min) && (count(Goal_i.Sat==“+S”) + count(Goal_i.Sat==“++S”)) <= OR.multiplicity.max)	$\text{Goal}_D.\text{Sat} = \max(\text{Goal}_i.\text{Sat})$

4 Morpheus: Using dynamic compilation techniques

With the aim of supporting this proposal a tool called MORPHEUS has been used. Due to the fact that it has to manage each model defined by ATRIUM (Requirements Model, Scenarios Model and Software Architecture Model) three different environments are provided by MORPHEUS (Fig. 4). Related to the Requirements Model, MORPHEUS is able not only to define both new kinds of artefacts and relationships but also to instantiate and exploit them, in such a way Goal Models with different expressiveness can be defined. For this reason, the Requirements Model environment has been split into two different working contexts. The first one allows analysts to establish the requirements metamodel to be used by means of the Metamodel Editor; and the second provides analysts with facilities for modelling according to the defined metamodel by using the Model Editor.

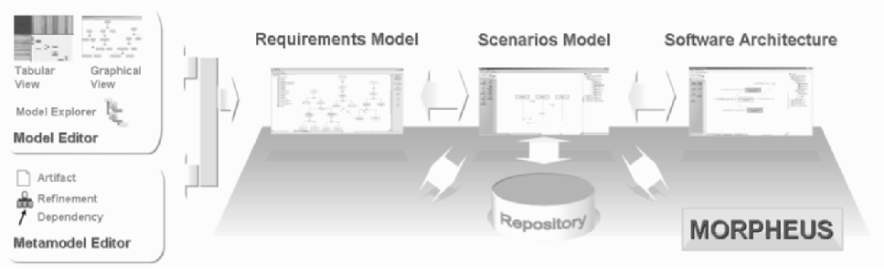

Fig. 4. Describing MORPHEUS capabilities MORPHEUS capabilities

In addition, MORPHEUS has been developed with capabilities to extend its functionality with analysis techniques. The main reason is that as new metamodels are defined, their related techniques can also be included and exploited. An example of this capability has been the development and integration of an add-in for Goal Model analysis based on satisfability propagation. Fig. 5 shows how this add-in has been designed. It has been split into three main components: a Rules Editor, a Code Compiler and a Propagation Processor.

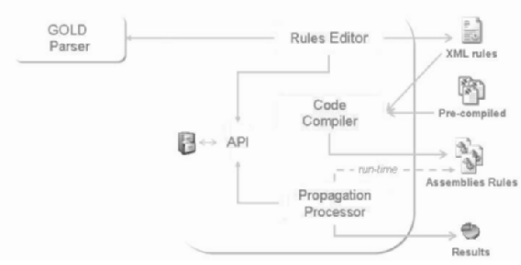

Fig. 5. A sketched view of the propagation add-in

Figure 6 shows what MORPHEUS looks like whenever the Rule Editor is loaded. For its development several alternatives were evaluated. However, the usability of the proposal was one of the main characteristics to be achieved. For this reason, a user interface (Rule Editor in Fig. 5) that allows the analyst to introduce the rules in a simple and comprehensible manner was developed. The Rule Editor is split into three main parts: a browser, a rules descriptor and an editor. The browser allows one to navigate through the relationships and the artefacts connected by them. The rules descriptor displays the applicable rules, for a selected relationship and source and destination artefacts. It can be appreciated that the *when* text box describes the condition and next to it appears the valuation. Below the rules descriptor, a visual control permits to edit the condition and the valuation. This provides the analyst with several buttons and capabilities that prevent him from knowing any detail about how his/her metamodel is described in the repository or how rules are internally implemented.

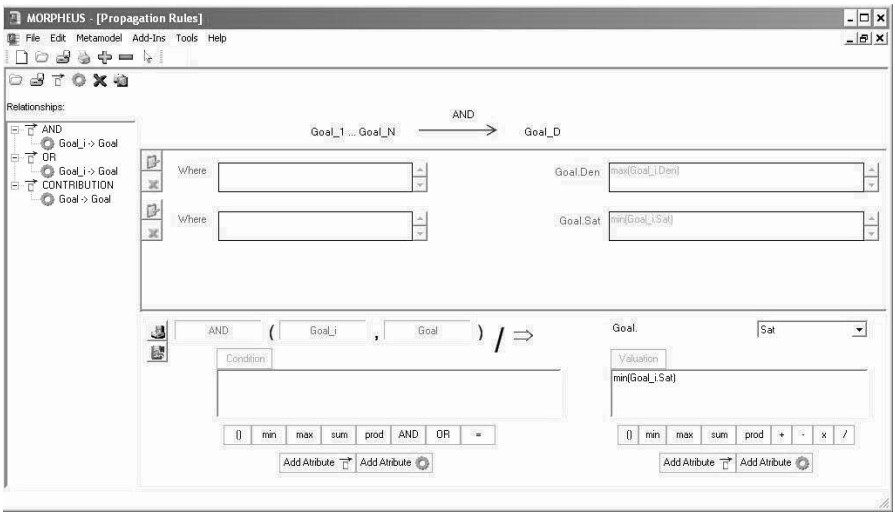

Fig. 6. MORPHEUS while loading the Rule Editor

In addition, a syntactic checking is performed when a rule is being defined thanks to the capabilities provided by (GOLD 2005). This is a free parsing system that can be used to develop one's own programming languages, scripting languages and interpreters by previously writing your grammar using BNF. The BNF, which was described for the condition (Table 2) and the valuation (Table 3), was introduced in GOLD. Then, the GOLD Parser Builder was used to analyze this grammar and create the Compiled Grammar Table file (CGT) used by a compilation engine. It uses this CGT file to generate a C# skeleton program with a custom parser class that acts as a template for parsing any source satisfying the BNF grammar. By means of this template, the specific compilation to the objective code can be described. In this case, a translation to C# code was performed to make available artifacts and relationships from the repository, and that computa-

tion which was needed for both condition and valuation. So, for each rule its valuation and condition description along with their respective compilation to C# is stored in a XML rules file to be lately used for the Code Compiler. The performance could have been seriously compromised due to the necessity of representing these rules as code for its use in run-time. For this reason, the approach of dynamic compiling, while it is more complex, provides us with a proper solution. Microsoft .NET Code Document Object Model technology, as described by (Harrison 2003), has been used for this purpose for the implementation of the Code Compiler.

By using Code Compiler (Fig. 5) a set of assemblies, containing both the rule code and other functionality, is generated at run time. For each rule, which is saved in a XML rule file, a C# class is generated which inherits from IRule. It is an abstract class with two abstracts methods to override for each inherited class: applicable function, which checks if the rule can be applied; and valuate function, which performs the propagation computation. This class also has a set of functions to perform the minimum, maximum, etc. So, while generating code, each rule C# class is going to override the abstract methods with that code stored in the XML rule file. Other classes are also used for the management of the generated classes which are previously pre-compiled to speed up this process.

Afterwards, these assemblies are accessed by the Propagation Processor to perform the propagation on a specific goal model and generate the results. In terms of integration, Propagation Processor makes use of the MORPHEUS API to access the model and will pass through the relations and artefacts retrieved from the repository.

5 Conclusions and further work

Goal Models are a very promising technique to improve requirements elicitation. Thanks to their special capabilities to analyze goals/requirements, they can be used at early stages of requirements engineering process, when alternatives are explored, conflicts are identified and, in general, the project is in the phase of requirement negotiation. However, Goal Model techniques must face a common obstacle in requirements engineering: the diversity of proposals with an evident lack of integration; and the specific needs of the project (or domain), which usually requires a customization of the requirement method and its notation.

We have presented a framework to cope with the integration and customization problem in requirements engineering techniques. In this work, we have used our framework to provide customizable support in Goal Model propagation analysis. We have illustrated how the propagation rules can be defined according to a specific metamodel. In this way, we can not only establish propagation rules for analysis but also redefine the propagation rules whenever it is necessary. This functionality was provided by means of dynamic compilation techniques using CodeDom. MORPHEUS, a tool we have developed to give support to our approach,

was extended with an add-in which allows the analyst to define propagation rules associated to the established metamodel and perform their computation.

Following this line of research, our ongoing work covers several issues. First, we intend to go more deeply into the analysis and exploitation of Goal Models using the specification of the goal/requirements itself apart from its attributes and relationships with other goals/requirements. This entails working with formal specifications of requirements (at least at some level of formalization) to perform a deeper verification and provide some automated support for the Goal Model specification. Other interest is to improve the mechanisms for analyzing the propagation, including graphical visualization (apart from tabular representation) and predefined report analysis. Finally, we are working on the establishment of precedence mechanisms for solving conflicts when more than one rule is applicable.

6 References

Chung L, Nixon B A, Yu E and Mylopoulos J (2000) Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing.

Dardenne A, Lamsweerde A van, and Fickas S (1993) Goal-directed Requirements Acquisition. Science of Computer Programming, 20, pp 3-50.

Giorgini P, Nicchiarelli E, Mylopoulous J, and Sebastiani R (2003) Formal reasoning techniques for goal models. Journal of Data Semantics, 1: 1-20.

GOLD Parsing System, http://www.devincook.com/GOLParser/ index.htm, 2005.

Hansen K M, Ravn A P, Stavridou V (1998) From Safety analysis to software requirements. IEEE Tran. on Software Engineering, 24(7):573-584.

Harrison N (2003) Using the CodeDOM. O'Reilly Network, http:// www.ondotnet.com/pub/a/dotnet/2003/02/03/codedom.html.

Kavakli E and Loucopoulos P (2004) Goal Driven Requirements Engineering: Analysis and Critique of Current Methods. In: Krogstie J, Halpin T and Siau K (eds) Information Modeling Methods and Methodologies, 102-124.

Lamsweerde A van (2000) Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. In: Proc 12th IEEE International Requirements Engineering Conference, IEEE Computer Society, Los Alamitos, pp 4-7.

Letier E and Lamsweerde A van (2004) Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. In: Taylor R N, Dwyer M B (eds) Proc of 12th ACM International Symposium on the Foundations of Software Engineering, ACM Press, New York, pp 53-62.

Navarro E, Letelier P, Mocholí, J.A. Ramos I (2006) A Metamodeling Approach for Requirements Specification. Journal of Computer Information Systems, 46(5): 67-77, Special Issue on Systems Analysis and Design.

Navarro E, Letelier P and Ramos I (2004) Goals and Quality Characteristics: Separating Concerns, Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, collocated to OOPSLA.

Navarro E, Ramos I and Pérez J (2003) Software Requirements for Architectured Systems. In Proc of 11th IEEE International Requirements Engineering Conference, IEEE Computer Society, Los Alamitos, pp 365-366 (position paper).