Model-Driven Development for Requirements Engineering: The Case of Goal-Oriented Approaches

Rui Monteiro, João Araújo, Vasco Amaral, Miguel Goulão, Pedro Patrício,
CITI/Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
{rm.chambel | pedropt}@gmail.pt, {vma | joao.araujo | mgoul}@fct.unl.pt

Abstract— **Goal-Oriented Requirements Engineering (GORE) has received increasing attention over the past few years. There are several goal-oriented approaches, each one using different kinds of models. We believe that it would be useful to relate them or even perform transformations among them automatically, in order to understand their similarities and differences, their advantages and disadvantages, allowing a possible migration or comparison between approaches. This is something that has not received enough attention. In this paper we propose the definition and implementation of goal model transformations between i* and KAOS. As an immediate contribution, the approach can be used to migrate from one goal model to another through automatic model transformations. This approach also contributes to relate the concepts of i* and KAOS models and will help, for example, a development team in making the decision on which approach to follow, according to the nature of the project and the expressiveness of an approach to represent certain concepts (e.g., obstacles are represented explicitly in KAOS, but not in i*). Another contribution is to facilitate communication among members of the same team, if they are specialized in different approaches.**

Keywords- Goal-Oriented Requirements Engineering, Model-Driven Development, Domain Specific Language

I. INTRODUCTION

Model-Driven Development (MDD) has a recognized role in software development in general due to its capabilities of producing more reliable software in a faster way. However, the main focus of MDD has been on design and implementation levels. But Requirements Engineering (RE) could also benefit from its advantages by allowing different kinds of requirements models transformations between different paradigms or inside the same paradigm. We are interested in studying the use of MDD in Goal-Oriented Requirements Engineering (GORE). GORE has a great impact and importance in the RE community [1]. It helps in identifying, organizing and structuring requirements, as well as in exploring and evaluating alternative solutions to a problem [1, 2]. In the context of GORE there is a wide variety of goal modeling languages, such as i* [3], KAOS (Knowledge Acquisition in AutOmated Specification) [4], GBRAM (Goal-Based Requirements Analysis Method) [15], GRL (Goal-oriented Requirement Language) [5] and NFR (Non-Functional Requirements) framework [6]. Despite the existing work on these approaches, such variety, according to [7], neither facilitates the selection of the appropriate approach, nor the mappings of models of different approaches. Thus, it is important to have a way to make it easier to learn the main differences of each one. It may be desirable to convert a model into another one because we may want to: refine an organizational level model (e.g. using i*models) to a system level model (e.g., by using a KAOS goal model); compare different approaches and decide which one is more expressive for a particular domain or to an organization culture (i.e. an approach can be more expressive than another to represent certain concepts — e.g. obstacles are represented explicitly in KAOS, but not in i*); facilitate the communication between professionals specialized in different approaches.

To achieve requirements model transformations, Model-Driven Development (MDD) [8] techniques can be used. These techniques range from metamodeling to transformation languages and tools (e.g., [9, 10, 11]). The final result is a more productive software development process. In this work, we propose a model transformation framework, called MDGore, to transform i* models into KAOS models through rules defined in ATL (Atlas Transformation Language) [9]. To achieve this, i* and KAOS models are specified using our tools that were developed in an Eclipse platform, LDE i* [12] and modularKAOS framework [13]. The application of the transformation rules is done at abstract syntax level, defined in the metamodel of each one of the frameworks mentioned, and uses the graph transformation paradigm [14], using the transformation language ATL. A poster version of this work was published with the very preliminary ideas [16]. Here we present a full-fledged approach with all the transformations implemented and applied to real-world case studies.

The paper is structured as follows. Section II provides a background on MDD and GORE. Section III presents a description of MDGore and its validation. Section IV discusses related work. Finally, Section V concludes the paper and gives directions for future work.

II. BACKGROUND

This section introduces the MDD and GORE. In GORE there is a wide variety of goal modeling languages, as mentioned in the previous section. i* and KAOS are among the most popular ones. As such, we chose them to discuss and illustrate the transformations between two GORE approaches.

A. Model-Driven Development

MDD [8] is a recent paradigm for software development, whose main idea is the systematic use of models and model transformations for the specification and implementation of software systems. The goal of MDD is to automate the process of creating new software and to facilitate its evolution in changing environments through model transformations [8]. In MDD, a model is typically associated with an abstract syntax that is defined by a metamodel. In other words, a model must be constructed following certain rules, and conform to a certain metamodel, which defines the model abstract syntax. In the same way, the metamodel is a model of a modeling language and must conform to the rules defined in the meta-metamodel. Once the metamodels and models are defined, the model transformations are used to derive a model from one or more models. That is, a model transformation takes as input a model conforming to a given metamodel and produces as output another model in conformance to a given metamodel. In the context of model transformation, our work uses the graph transformation approach. A graph transformation [14] is a technique for model transformation that involves models that can be represented as graphs, which is the case of i* and KAOS models. Graph transformation rules consist of a LHS (Left-Hand Side) graph pattern and a RHS (Right-Hand Side) graph pattern [14], which correspond to the input and output model elements respectively. The LHS pattern is matched against the model being transformed and the RHS pattern is generated as the output model.

Model transformation languages [9, 10, 11] aim at automating the process of deriving one model from another one. We have chosen ATL [9] for specifying our model transformations, since it offers the possibility to define the rules in a hybrid way, that is, in a declarative or imperative programming style [9]. The preferred style of transformation writing is the declarative one. This style enables to simply express mappings between the source and target model elements. The imperative constructs can be used when mappings are difficult to specify in the declarative style. An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models [9].

B. The i* approach

i* [3] was developed for modeling and reasoning about organizational environments and their information systems [3]. It focuses on the concept of intentional actor. Actors in their organizational environment are viewed as having intentional properties such as goals, beliefs, abilities and commitments. i* has two main modeling components: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model describes the dependency relationships among the actors in an organizational context. In this model, an actor (called depender) depends on another actor (called dependee) to achieve goals and softgoals, to perform tasks and to obtain resources. The SR model provides a more detailed level of modeling than the SD model, since it focuses on the modeling of intentional elements and relationships internal to actors. Intentional elements (goals, softgoals, tasks and resources) are related by means-end or decomposition links. Means-end links are used to link goals (ends) to tasks (means) in order to specify alternative ways to achieve goals. Decomposition links are used to decompose tasks. A task can be decomposed into four types of elements: a subgoal, a subtask, a resource, and/or a softgoal. Apart from these two links, there are the contribution links, which can be positive or negative.

In this work we use LDE i* [12], a Domain Specific Language (DSL) [17] for i*. Its main objective is to ensure a better management of complexity and scalability of their models by introducing the notion of compartment where elements of an i* model can be grouped. This DSL was implemented using MDD techniques, and includes an Ecore [19] metamodel based on existing ones [3, 18]. Figure 1 shows a fragment of the LDE i* metamodel. The full version can be found in [12]. The root of the metamodel is the class SRModel. This class is composed of zero or more nodes, represented by the abstract class Node, and zero or more links, represented by the abstract class Relationship. A node can be a DependableNode type, specialized as an Actor, or the Dependum type. A Dependum may be one of the following types: Task, Resource, Goal, SoftGoal, ElementContainer and SoftGoalContainer. The latter two represent the compartments. The ElementContainer compartment groups the Tasks, Goals and Resources. The SoftGoalContainer compartment groups the SoftGoals. A relationship can be one of the following types: ContributionLink, DecompositionLink, MeansEndLink and Link. The Link class represents the dependency relationship through classes DependeeLink and DependerLink. This DSL was implemented using the EMF/GMF [19, 20], which are plug-ins of Eclipse.

C. The KAOS approach

KAOS [4] is a systematic approach for discovering and structuring system level requirements. In KAOS, goals can be divided into requirements (a type of goal to be achieved by a software agent), expectations (a type of goal to be achieved by an environment agent) and softgoals (e.g., quality attributes). In KAOS, goals can be refined into subgoals through *and/or* decompositions. There is also the possibility of specifying conflicts between goals. KAOS also introduces the concept of obstacle that is a situation that prevents the achievement of a goal. Usually the solution to the obstacle is expressed in the form of a new requirement.

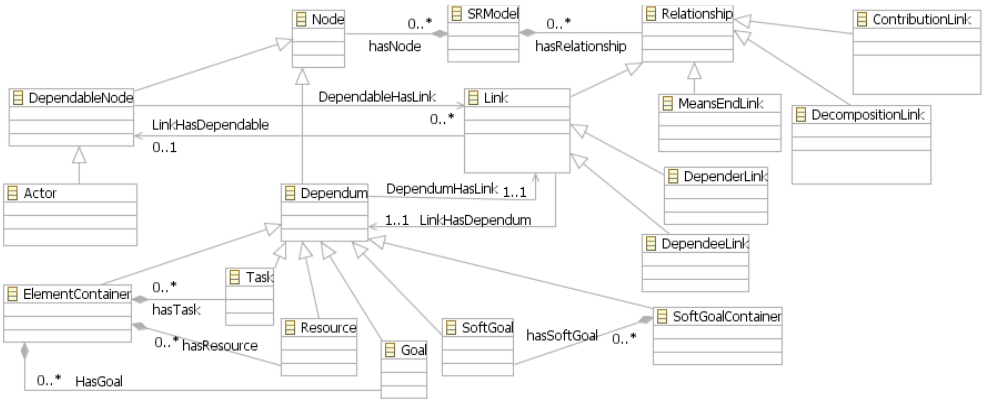

Figure 1. Partial LDE i* metamodel.

Here we use modularKAOS [13], which is a DSL, implemented using MDD techniques (metamodeling), for dealing with KAOS models. This DSL also incorporates the notion of compartment, in KAOS models, to improve their scalability. Each compartment can contain elements of the KAOS model and can be collapsed to hide what is inside it. This feature can be useful to help analyzing just a portion of the model, while abstracting others. This DSL was implemented based on a metamodel defined in [21], using Ecore [19], extended with the concept of compartment. Figure 2 shows a partial modularKAOS metamodel. The full version can be found in [13]. The class KAOS is the metamodel root. This class is composed of zero or more CompartmentNode, Nodes and Links.

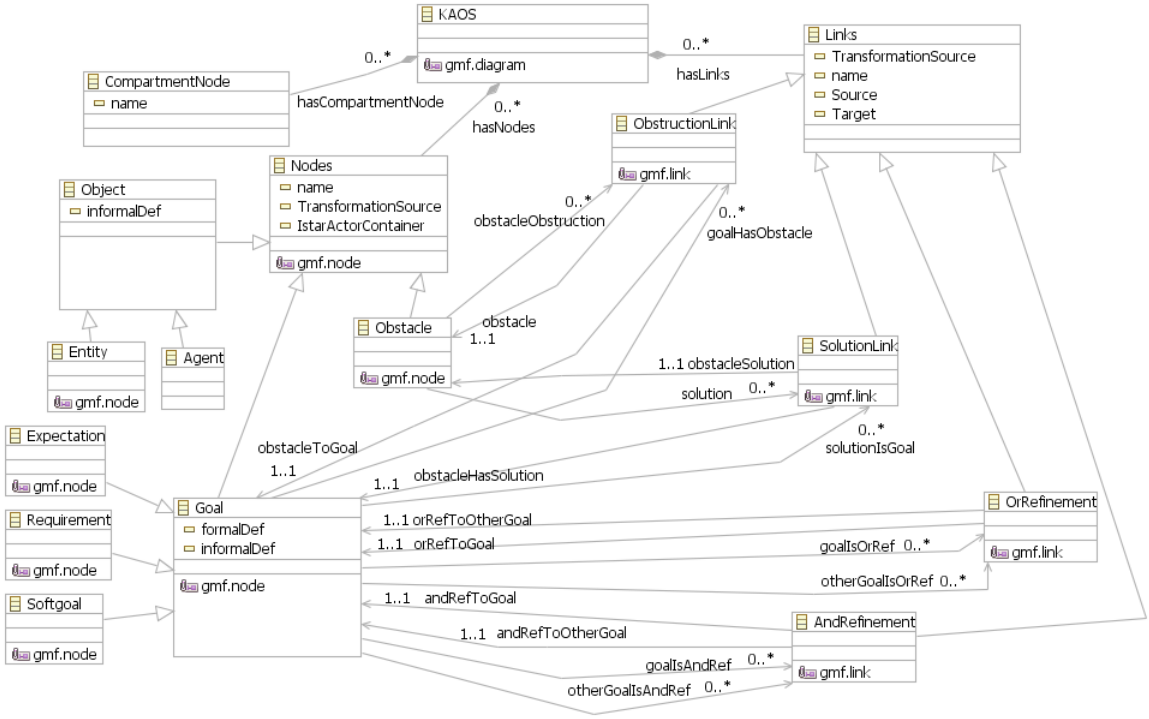

Figure 2. Partial modularKAOS metamodel.

The Nodes class represents the nodes that can be Object, Goal or Obstacle. An Object can be of the Agent or Entity type. A Goal can be one of the following types: Expectation, Requirement or Softgoal. Links can be: ObstructionLink, SolutionLink, OrRefinement or AndRefinement. The link ObstructionLink connects an Obstacle to a Goal and a link SolutionLink connects a Goal to an Obstacle. Links OrRefinement and AndRefinement refine Goals. This DSL was implemented similarly to LDE i*.

III. MDGORE: A MODEL-DRIVEN GOAL-ORIENTED REQUIREMENTS FRAMEWORK FOR I* AND KAOS

This work proposes MDGore, a model-to-model transformation approach between i* and KAOS models. This transformation consists in transforming i* SR into KAOS Goal models. Figure 3 gives an overview of our framework. The dashed rectangle is the focus of our approach. The source model is the i* SR model which is transformed into a KAOS Goal model through the application of rules implemented in ATL.

An i* SR model is specified using the LDE i* framework and it conforms to the i* metamodel implemented in this framework. A KAOS model conforms to the KAOS metamodel implemented in the DSL of modularKAOS. ATL rules conform to the ATL metamodel. Finally, these three metamodels were implemented using the Ecore formalism [19] and, therefore, conform to the Ecore meta-metamodel. This framework is semi-automatic, since the initial phase of the transformation process is applied for user intervention, to make decisions about the mapping of some i* elements into the corresponding KAOS elements. These specific cases will be addressed in the next section. We defined a metamodel, illustrated in Figure 4, to generate a "decision model" so that the users can make and record their choices.

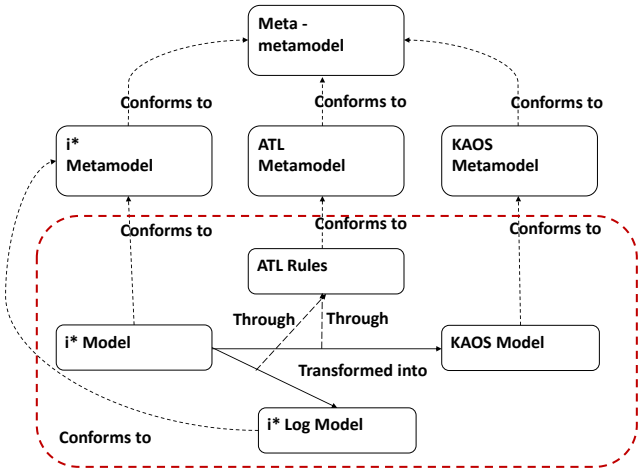

Figure 3. Process overview of the MDGore framework.

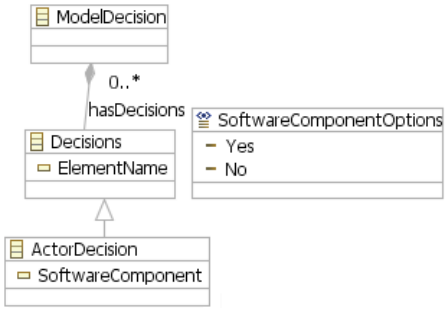

Figure 4. Metamodel of the decision model.

The root of this metamodel is the ModelDecision class, which is composed of zero or more decisions, represented by the abstract class Decisions. This class represents an abstract decision element and has the attribute ElementName of String type that corresponds to its name. The ActorDecision class represents the i* element in which a decision will be recorded. This class extends the Decisions class and has an attribute, SoftwareComponent, of Enumeration type. The Enumeration, called SoftwareComponentOptions, has the values "Yes" or "No". The decision model is generated from the i* SR model and contains elements of type ActorDecision under which decisions are made. For each of these elements the user must decide which option, on the attribute SoftwareComponent, is more appropriate.

One of the problems associated with the transformations between models of different approaches is the information loss. This problem arises from the fact that the abstract syntaxes of the approaches involved in the transformations are different and, as a consequence, it is not always possible to transform all the elements. This problem is also present when relating i* and KAOS models. To address this problem we introduce the notion of a Log model. This model keeps the information of all the elements that were not possible to transform. This model is represented using the i* notation. Looking again at Figure 1, the application of the transformation rules results in two models: one KAOS Goal model with the transformed elements and a Log model, with the non-transformed elements of the i* SR model.

Another concern that we had was to ensure traceability in the transformation process. Through attributes in the KAOS metamodel, we store the history of the transformation in the transformed KAOS model. For example, when an i* element is transformed into a KAOS element, the type of the i* element is stored into an attribute of the KAOS element. This solution enables the possibility of transforming a KAOS Goal model back to the corresponding i* SR model, ensuring bidirectionality in the transformation process.

Finally, our framework is flexible enough to change the transformed model, in particular by removing, adding, or changing, elements in the model. To implement the transformation rules, a study of relations between the elements of the two selected approaches was made, in order to establish their mappings. This is discussed next.

A. Mapping between i and KAOS elements*

Here we present the relations between i* and KAOS elements. These relationships have been established based on the concepts of the two approaches introduced in Section 2 and based on the abstract syntax of each approach. The KAOS abstract syntax is defined in the modularKAOS metamodel that was based on the metamodel defined in [21]. In turn, the i* abstract syntax is defined in the LDE i* metamodel, which was based on the metamodel described in [3, 18]. Table I represents the mappings of i* model elements into KAOS model elements.

As shown in Table 1, a task can be mapped to an expectation or a requirement, depending on the mapping of the actor that contains it. That is, if the actor is mapped to an EnvironmentAgent the task is mapped to an expectation, whereas if the actor is mapped to a SystemAgent the task is mapped to a requirement.

The reason for this type of mapping is related to the restrictions imposed by KAOS. In KAOS, an expectation can only be associated with EnvironmentAgents, as well as a requirement can only be associated with SystemAgents. The i* actor types (actor, position, agent and role) can be mapped to SystemAgents or EnvironmentAgents. This is the only mapping that cannot be done automatically, since application domain knowledge is needed. Here, user intervention is necessary to select, through the decision model, the type of KAOS agent that corresponds the type of i* actor. The decomposition link can be mapped to an AndRefinement link or concernsLink.

TABLE I. MAPPINGS OF I* ELEMENTS INTO KAOS ELEMENTS

i*	KAOS
Goal	Goal
Task	Expectation; Requirement
Softgoal	Softgoal
Resource	Entity
Actor, Position, Agent, Role	SystemAgent; EnvironmentAgent
Decomposition	AndRefinement; ConcernsLink
Means–End Link	AndRefinement; OrRefinement
Is-part-of Association	Aggregation
ISA Association	Inheritance
Break, Some-, Hurt	Conflict
Make, Some+, Help	OrRefinement
Dependency	AndRefinement

If a task is decomposed into subtasks, or goals, or softgoals, the decomposition link is mapped to an AndRefinement. If a task is decomposed into resources, the decomposition link is mapped to a concernsLink. This happens because an i* resource is mapped into a KAOS entity, and in KAOS the links that involve entities are concernsLinks. Finally, the means-end link has two possible mappings: AndRefinement or OrRefinement. If the source element of the means-end link has another means-end link, each of the destination elements is seen as an alternative to obtain the same source element. In this case, each means-end link is mapped to an OrRefinement. On the other hand, if the source element has only one means-end link, its target element is the only way to get it and this link is mapped to an AndRefinement. Regarding the dependency link, this is only transformed if the relationship occurs between the internal elements of the i* actors. If the dependency link is between actors it cannot be transformed, since there is no correspondence in KAOS. This dependency link between actors is the only i* element that is not possible to transform directly, so user's intervention is needed.

B. *Example of transformation*

To help explaining the transformation process we will introduce a partial i* SR model (illustrated in Figure 5) of the "Project BTW: if you go, my advice to you" [22]. The main goal of this project is to develop a route planning system that allows community input. The model contains two actors: Travelers and Internet.

The goal of the Travelers is to have a *Trip to interesting places* realized. The Task *Travel to interesting places* is a means to achieve that goal. This task is decomposed into the *Planning trip* subtask. To plan a trip it is necessary to perform the following subtasks: *Get destination info*, and (select) *Transport to destination*. The goal of the Internet is to have Services provided. The task *Provide services* is a means to achieve that goal. This task is decomposed into the following subtasks: *Provide transport info* and *Provide places info*. To provide info about transport, this subtask requires the resource *Transport info*. Finally, the actor Travelers depends on the actor Internet to achieve the goal *Information be provided*.

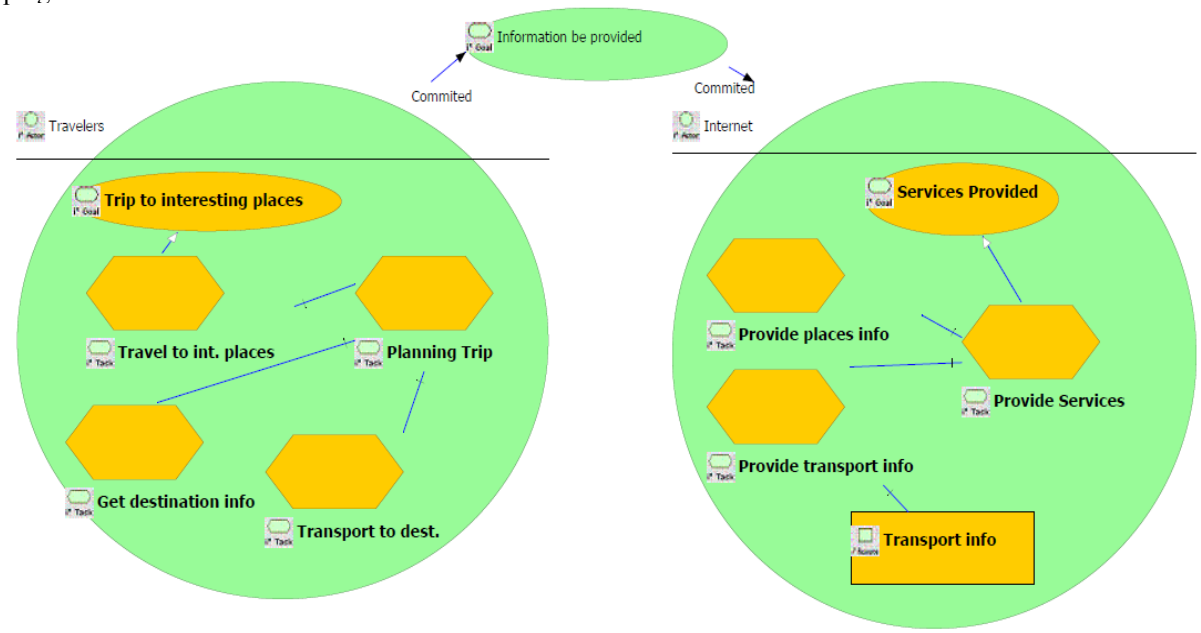

Figure 5. Partial i* SR model of the BTW Project

C. *Transformation Process*

In this subsection we will describe the transformation process of an SR model into a KAOS Goal model. This transformation process is based on the mappings described previously, and consists of: (i) generation of a decision model; (ii) generation of an annotated SR model; (iii) generation of an intermediate KAOS Goal model; (iv) generation of the final KAOS Goal model; and (v) generation of a Log model. For each step a set of ATL transformation rules was specified and implemented.

Step 1: Generation of a decision model. In this step a decision model from the SR model is generated, so that the user can infer decisions about the type of some i* elements, based on domain knowledge of the problem. As mentioned, the four types of i* actors can be mapped into two types of KAOS agents: SystemAgent or EnvironmentAgent. It is up to the user to decide which agent should map to. A decision model is presented (Figure 6), containing the actors of the i* SR model.

In this decision model, for each actor, the user must select whether it is a software component by using the options "Yes" or "No" of the property SoftwareComponent. The "Yes" option means that the actor is part of the system and in this case it will then be mapped to a SystemAgent. The "No" option means that the actor is an intervening one in the system and will then be mapped to an Environment Agent.

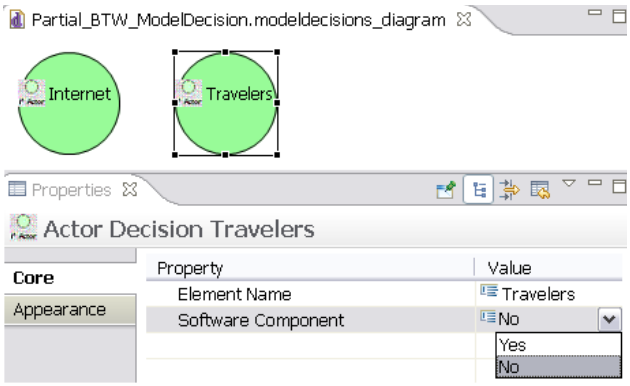

Figure 6. Decision model example

In the example, for the Travelers actor it was selected the "No" option in the property SoftwareComponent, which means it will be mapped into an EnvironmentAgent. Although Figure 6 does not show, for the Internet actor it was selected the "Yes" option, which means that it will be mapped to a SystemAgent.

Step 2: Generation of an annotated i* SR model. After taking all the decisions in the previous step, this step generates an annotated SR model, from the decision model and the original SR model, with all these decisions. In fact, the annotated model contains the same elements as the original SR model, and is enriched with user decisions. From this step all the mappings are defined and the conditions are satisfied to perform the transformation into an intermediate KAOS Goal model.

Step 3: Generation of an intermediate KAOS Goal model. In this step, an intermediate KAOS Goal model is generated from the annotated SR model of the previous step. This model is intermediate because KAOS has responsibility links between the expectations or requirements and their agents that do not exist in i*. As a consequence, there is no i* element to serve as a pattern for generating the responsibility link. The solution was: an i* element generates two KAOS elements, one being the responsibility link and the other a task. However, this solution does not fully solve the problem, since the model is poorly structured. This problem is resolved in the next step.

We selected 3 rules to illustrate the transformations we defined and implemented. The rules in Listings 1, 2 and 3 specify how some i* elements are transformed into KAOS elements. These rules transform the pattern Means-End Link between a goal and a task into the pattern AndRefinement between a goal and an expectation. With the rule in Listing 1 we can transform the i* goal Trip to interesting places into the KAOS goal with the same name. The *from* clause corresponds to the input pattern (LHS pattern according to graphs transformation technique). The variable *i* represents the i* goal. The *to* clause corresponds to the output pattern (RHS pattern) which is generated by the input pattern set in the from clause. The variable *out* represents the KAOS goal that will be obtained from the i* goal defined in the variable *i*. The attribute *name* of the KAOS goal is assigned with the name of the i* goal. The attribute *TransformationSource* is filled with the type of i* element to store the transformation history.

```
rule IstarGoal2KaosGoal {
    from i : iStarPrototipe!Goal (i.TransformationSource =
        'Goal' or i.TransformationSource='' ' or
i.TransformationSource.oclIsUndefined())
    to out : KAOS!Goal (
            name <- i.Name,
            TransformationSource <- 'Goal') }
```

Listing 1. ATL rule to transform i* Goal into KAOS Goal.

```
rule IstarITask2KaosExpectation {
    from i : iStarPrototipe!ITask
        ( (thisModule.getActor(i).TransformationSource =
'EnvironmentAgent' and
        (i.TransformationSource = '' or
i.TransformationSource.oclIsUndefined() ) )  or
        (i.TransformationSource = 'Expectation') )
    to out : KAOS!Expectation (
            name <- i.Name,
            TransformationSource <- 'ITask')}
```

Listing 2. ATL rule to map i* Task into KAOS Expectation.

With the rule in Listing 2 we can transform the i* task *Travel to interesting places* into the KAOS expectation with

the same name. The description is similar to the previous one where the variable *i* represents the i* task, the variable *out* represents the KAOS expectation that will be transformed from the i* task set in the variable *i*. The name of the i* task is assigned to the attribute name of the KAOS expectation. With the rule in Listing 3 we can transform the i* means-end link, between the previous i* goal and the i* task, into the KAOS AndRefinement link. In the *from* clause the variable *i* represents the i* means-end link. In the *to* clause the variable *out* represents the KAOS AndRefinement link that will be transformed from the i* means-end link set in the variable *i*. The attribute andRefToGoal of the KAOS AndRefinement corresponds to the source link and is assigned the source link of the i* means-end link. The attribute andRefToOtherGoal of the KAOS AndRefinement corresponds to the target link and is assigned the target link of the i* means-end link. The remaining elements are transformed following the same procedure.

```
rule IstarMeansEndLink2AndRefinement {
    from i : iStarPrototipe!MeansEndLink
        (thisModule.MoreThan1MeansEndLink(i.MeansEndHasGoal) < 2)
    to out : KAOS!AndRefinement(
            andRefToGoal <- i.MeansEndHasTask,
            andRefToOtherGoal <- i.MeansEndHasGoal,
            TransformationSource <- 'MeansEndLink')}
```

Listing 3. ATL rule to map i* Means-End Link into KAOS AndRefinement

Step 4: Generation of the final KAOS Goal model. The output of this step is a final KAOS goal model, illustrated in Figure 7, which is generated from the intermediate KAOS goal model of the previous step. The purpose of this step is to complete the KAOS goal model structure generated in the previous step. In this sense, it was created an ATL rule to restore the responsibility links between the elements and their agents. These responsibility links are visible in the Figure 7, between the EnvironmentAgent Travelers and their expectations and between the SystemAgent Internet and its requirements.

Step 5: Generation of a Log model. The purpose is to generate a Log model from the annotated SR model in the second step, with the i* elements that were possible to transform. However, it may appear in the Log model elements that have been transformed, but were related to elements that are not able to be transformed, which is the case of some relationships. For example, in the SR model of Figure 5, the dependency relationship between actors is not possible to be transformed. This relationship is stored in the Log model, shown in Figure 8, together with the actors (Travelers and Internet) and the goal (Information provided) in the center of the dependency. Despite these elements (actors and goal) have been transformed, they appear in the Log model because they are part of the dependency link, that is not able to be transformed, and because it is not possible to store such a link without its source and target elements. If all the SR model elements are able to be transformed, an empty Log model is generated. Otherwise, a Log model is generated with the elements that are not transformed.

D. Transformation result

The output of the transformation process explained above is a KAOS goal model with the transformed elements, and a Log model with the i* elements that have not been possible to be transformed. In the KAOS goal model, illustrated in Figure 7, we can see that the i* goals were transformed into KAOS goals. The Travelers actor was transformed into an EnvironmentAgent, as in the decision model we decided that it is not a software component. For the Internet actor, in the same decision model, we decided that this actor is a software component and in this case it was transformed into a SystemAgent. The internal tasks of the Travelers actor were transformed into expectations since this actor corresponds to an EnvironmentAgent. Also, the internal tasks of the Internet actor were transformed into requirements, since this actor corresponds to a SystemAgent. Internal resources to the Internet actor were transformed into entities.

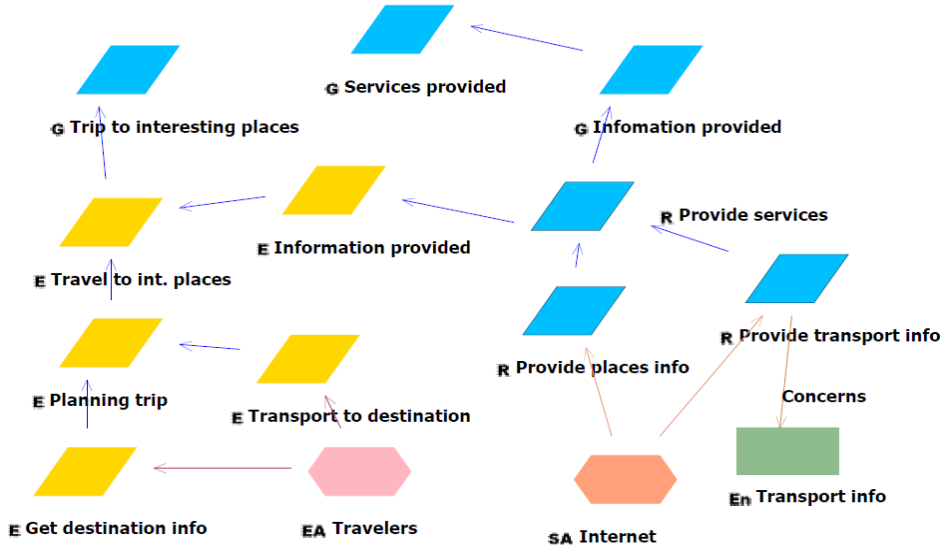

Figure 7. KAOS Goal model obtained.

All decomposition links between tasks were transformed into AndRefinement links and decomposition links between tasks and resources were transformed into ConcernsLinks. The means-end link between the goals and tasks were transformed into AndRefinement, as each goal has only one means-end link.

As we mentioned, our framework is flexible, allowing users to change the KAOS Goal model generated by the transformations. This is suitable since it is not possible to automatically transform the dependency relationship between actors (Figure 8).

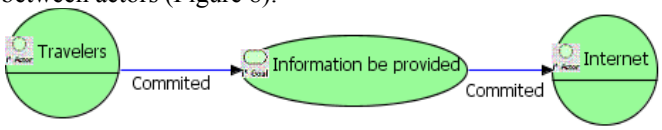

Figure 8. Log model.

So we changed the KAOS Goal model in order to establish a possible solution to this dependency relationship. The solution was to represent the dependum Information provided for both perspectives, i.e., Travelers' and Internet's. Firstly, we specified the expectation *Information provided* (Traveler's perspective) resulting from the refinement of the expectation *Travel to interesting places*. Secondly, the expectation *Information provided* is refined into the requirement *Provide services*, which is under the responsibility of Internet SystemAgent, and as such depends on this requirement to be fulfilled. We also created the goal *Information provided* (Internet's perspective) which is refined into the requirement *Provide services*, depending on this to be achieved. Thus, it is possible to establish the dependency relationship between Travelers and the Internet, through the dependency between the expectation and requirement.

E. Validation

As a proof of concept, we applied the approach to the Health Watcher case study. Health Watcher is a real-world system that provides information about public health [23]. By modeling it with the approach, we assessed our transformation rules correctness with a real-world case study. The detailed model of the Health Watcher system built with our approach can be found in [28].

Concerning the effect of the proposed approach and its tool support in Goal-Oriented Requirements Engineering activities, we conducted a pilot study with a group of 7 final year MSc students in Computer Science at [University name removed for the sake of the double blind review process]. These graduate students had previously taken courses on Requirements Engineering and on Domain-Specific Languages Development, so they had enough previous experience with GORE and MDD, as they had applied both to several case studies in the context of those courses. None of the participants was working under the supervision of the authors, or enrolled in courses taught by the authors, to minimize potential biases that could otherwise occur.

The participants received 60 minutes training. After completing their training, they were asked to perform a set of modeling tasks. First, they created manually a KAOS Goal model based on the SR model of the BTW Project [22]. Then, using the approach, they transformed the SR model into the KAOS goal model and compared it with the Goal model they created manually. Finally, participants were asked to fill in a questionnaire to compare the approach with the baseline manual approach.

The questionnaire included three questions:

- Question 1: *"How do you rate the model generated by the transformations in comparison to the model created manually?"(1 – Very bad; 2 – Bad; 3 – Similar 4 – Good; 5 – Very good)*
- Question 2: *"Was there any information lost in the transformation process?" (1 – Yes; 5 – No)*
- Question 3: *"How do you rate the simplicity of the transformation process?"(1 – Complex; 2 – Little Simple; 3 – Reasonably Simple; 4 – Simple; 5 – Very Simple)*
- Question 4: *"How do you rate the usefulness of the approach?" (1 –Useless; 2 – Little Useful; 3 – Somehow useful; 4 – Useful; 5 – Very Useful)*

Table II summarizes the frequencies of responses, broken down by question. Note that we only have 2 categories for question 2, while we have 5 for the remaining questions.

TABLE II. QUESTIONNAIRE'S ANSWERS FREQUENCY, BY QUESTION

Category	Question 1	Question 2	Question 3	Question 4
1	0	0	0	0
2	0	0	1	0
3	0	0	5	0
4	2	0	1	1
5	5	7	0	6

Overall we can observe higher frequencies in the categories implying some sort of success, for all but question 3, where the results are neutral (meaning that most participants found the transformation process only reasonably simple). Assuming that answers falling into categories 1 and 2 mean that our approach is worse than the baseline, category 3 means that our approach is as good as the baseline, and categories 4 and 5 mean that our approach is an improvement from the baseline, we aggregated the answers into *"No improvement"* (categories 1, 2 and 3) and *"Improvement"* (categories 4 and 5). The aggregated frequencies are presented in table III.

TABLE III. AGGREGATED RESULTS, BY QUESTION

Category	Question 1	Question 2	Question 3	Question 3
No improvement	0	0	6	0
Improvement	7	7	1	7

These aggregated results show improvements on questions 1, 2 and 4. For each of these questions, all participants chose the top categories. 5 out of 7 participants indicated that the model generated by transformations compared with the model they created was *"Very good"*. Only two users rated it as *"Good"*, which is still a positive rating. None of the participants reported any information loss with the approach, so this objective was fully met. 6 out of 7

participants rated the approach as *"Very useful"*, while the remaining participant rated it as *"Useful"*. The participants particularly enjoyed the support for interoperability among teams with different expertise in GORE, which was one of the planned contributions of this approach.

Concerning question 3, the answers were mostly neutral (5 out of 7 considered the transformation process to be *"Reasonably Simple"*). In contrast with the remaining questions, the simplicity of the approach was still considered a challenge with the proposed approach. We further explored this and found that the participants considered that the settings of the transformation process were not as simple as they would feel comfortable with. They suggested that the creation of a wizard in the tool support would mitigate this problem.

Overall, these results are encouraging and helped us identifying an opportunity for improving the tool support, through the inclusion of the wizard for setting up the transformation process, as suggested by our participants. We performed a Chi-Square test on the distributions of the collected answers (both detailed and aggregated – only detailed for question 2, because it has only two possible answers). Table IV presents the test results with the Chi-Square statistic, the number of degrees of freedom (Df) and the asymptotic significance. Although the asymptotic significance in all but the aggregated Question 3 results might lead us to conclude that the results are statistically significant, we must stress that the number of participants in this assessment was relatively small, with only 7 participants. As such, a replication of this evaluation should be performed before more definitive conclusions can be drawn in a sound way. To facilitate such replication efforts, an experimental package is available at (link omitted for the sake of the double blind review – the authors can make it available through the PC chair, if necessary for the paper review). Nevertheless, it is common for usability tests to be performed by a relatively small number of users, and still being considered meaningful, even if they are not statistically significant. Usability experts have noted that a group of 5 testers is enough to uncover over 80% of the usability problems [24]. As our questionnaire is focused on the usability of the approach, 7 participants is a reasonable number, from a pragmatic point of view, for this first assessment, at least with respect to identifying challenges on the usability of the approach.

TABLE IV. CHI-SQUARE TESTS STATISTICS

Questions:	**Q 1**		**Q 2**	**Q 3**		**Q 4**	
	Detail.	Aggr.	Detail.	Detail.	Aggr.	Detail.	Aggr.
Chi-square	13,714	7,000	7,000	12,186	3,571	19,429	7,000
Df	4	1	1	4	1	4	1
Asymp. Sig.	0,008	0,008	0,008	0,015	0,059	0,001	0,008

Concerning external validity threats, we only used one case study due to limitations on participant's availability. However, the chosen case study is significantly complex for evaluating the more intricate cases. Another threat to external validity is the fact that all participants were graduate students with little professional experience thus limiting our ability to generalize the results of validation. However, the knowledge level of the students can be considered adequate and even higher when compared to young software engineers in industry, in particular with respect to GORE. Using graduate students as surrogates for young professionals is a common approach in experimental software engineering [2].

IV. RELATED WORK

In [25] a framework is proposed for tracing aspects from requirements goal models to implementation. The framework provides language support for modeling goal aspects and mechanisms for transforming models to aspect-oriented programs. The main difference between this approach and ours is that it uses model-to-code transformation, while our approach uses model-to-model transformation.

[26] presents a semi-automatic approach that aims to derive an aspect-oriented (AO) architecture from an AO requirements specification. An AO requirements scenario model is automatically transformed into an AO architectural model. The transformations are implemented in ATL. Transformations are made to different levels of abstraction of software development – from requirements to architecture-while in our approach they are made at requirements level.

The proposal presented in [27] describes a UML AO modeling tool, called MATA, which uses graph transformations to specify and compose aspects. Its main goal is to compose aspects in UML models. MATA currently supports composition for UML class, sequence and state diagrams. In contrast with our approach MATA graph rules are defined over the concrete syntax of the modeling language, in this case the UML. Comparing this approach with ours, MATA rules are defined in terms of concrete syntax, our approach's rules are defined using abstract syntax.

In [29], the authors present an algorithm to automatically generate UML statecharts from a collection of scenarios represented using UML sequence models. In this work, they address several issues, such as detecting conflicts arising from the merging of independently developed sequence models and find behavioural similarities between different sequence models. They do this at the algorithm or transformation level. In [30] the authors propose to generate automatically, through model transformations, an activity model representing the use case scenario from a textual template. In this work, we observed that the semantics inherent to the abstractions present on the template (if-then-else, requirements numeration indicating parallelism) and on the activity model were very close, which resulted in a relatively trivial set of transformation rules. These approaches are focused on behavioral models' transformations while our approach is concerned with goal models' transformations.

In [31] it is reported the current state of MDD approaches with respect to NFRs. The authors say that in general, NFRs are not addressed in MDD methods and processes. They outline a general framework that integrates NFRs into the core of the MDD process, however no full-fledged approach was proposed.

In [32], we have proposed a new extensible language to unify and represent GORE languages. The unifying treatment of concepts in our model makes it possible to define more projections of each problem than the existing languages by themselves. Also hybrid models can be used. The proposed language provides mapping between syntactic constructs across languages, but cannot bridge the difference in semantics. Each language was developed based on specific theories. For example, i* was developed based on the resource dependency theory, but i* cannot just add constructs such as obstacle from KAOS. This is the reason why we introduced the idea of hybrid use of multiple languages. However, if the adoption of a unifying language is not possible, a transformation based approach, proposed in this paper is a plausible alternative.

V. CONCLUSIONS

In this paper, we have expanded a preliminary work [16] by producing a complete semi-automatic model-to-model transformation framework, to relate KAOS and i* models. For these models, we defined and implemented all the transformation rules (complementing the preliminary work), defined at the abstract syntax level, using ATL. We validated our approach by applying it to two case studies and via a questionnaire filled by a group of MSc students with knowledge on GORE and MDD. The validation process collected positive results and evidence to confirm the framework's usefulness.

Generally, by doing this work we observed that MDD can be used successfully to relate models of different approaches, in different abstraction levels, that share the same paradigm. Furthermore, the MDGore approach brings the following contributions to the requirements community: refinement of an organizational level model (e.g. using i*models) to a system level model (e.g. by using a KAOS goal model); help deciding which approach is more expressive than another to represent certain concepts — e.g. obstacles in KAOS, which does not exist in i* explicitly; facilitate the communication between professionals specialized in different approaches.

Our future work includes the developing transformations to others requirements approaches using different paradigms. Finally, we have already concluded the transformations from KAOS to i*, which will be object of a future submission.

REFERENCES

[1] Lamsweerde, A. 2001. Goal-Oriented Requirements Engineering: A Guide Tour, RE'01, Canada.

[2] Regev, G., Wegmann, A. 2005. Where do Goals Come from: the Underlying Principles of Goal-Oriented Requirements Engineering, RE'05, France.

[3] Yu, E. 1995. "Modelling Strategic Relationships for Process Reengineering", PhD thesis, University of Toronto, Canada.

[4] Lamsweerde, A. 2009. Requirements Engineering: From System Goals to UML Models to Software Specifications, Willey.

[5] GRL Webpage, http://www.cs.utoronto.ca/km/GRL/.

[6] Chung, L., Nixon, B., Yu, E., Mylopoulos, J. 2000. NFR in Software Engineering: Kluwer, 2000.

[7] Matulevičius, R., Heymans, P. 2007. Comparing Goal Modelling Languages: an Experiment, REFSQ'07, Norway.

[8] Völter, M., Stahl, T. 2006. Model-Driven Software Development - Technology, Engineering, Management, Wiley.

[9] ATL User Guide: http://wiki.eclipse.org/ATL/User_Guide.

[10] QVT, Webpage: http://www.omg.org/

[11] VIATRA2 http://www.eclipse.org/gmt/VIATRA2/

[12] Nunes, C., Araújo, J., Amaral, V., Silva, C. 2009. A Domain Specific Language for the i* Framework, ICEIS'09, Italy.

[13] Dias, A., Amaral, V., Araújo, J. 2009. Towards a Domain Specific Language for a Goal-Oriented Approach based on KAOS, RCIS'09, Morocco.

[14] Czarnecki, K., Helsen, S. 2003. Classification of model transformation approaches, Workshop on Generative Techniques in the context of MDA, USA.

[15] Antón, A. 1996. Goal-Based Requirements Analysis, ICRE'96, USA.

[16] Monteiro, R., Araújo, J., Amaral, V., Patrício, P. 2010. Mdgore: Towards Model-Driven and Goal-Oriented Requirements Engineering, RE'10, Australia.

[17] Kelly, S., Tolvanen, J. 2008. Domain Specific Modeling, Wiley.

[18] Alencar, F., Silva, C., Lucena, M., Castro, J., Santos, E., Ramos, R. 2008. Improving the understandability of i* models, ICEIS'08, Spain.

[19] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. 2008. EMF Eclipse Modeling Framework, Addison-Wesley.

[20] The GMF Webpage: http://www.eclipse.org/modeling/gmf/.

[21] Matulevičius, R., Heymans, P. 2005. Analysis of KAOS Meta-model, Technical report, University of Namur, Belgium.

[22] Lucena, M., Castro, J., Silva, C., Alencar, F., Santos, E., Pimentel, J. 2009. A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models, OTM Workshops.

[23] Massoni, T., Soares, S., Borba, P. 2007. Requirements Health-Watcher version 2.0, Early Aspects, ICSE'07, USA.

[24] Nielsen, J., and Landauer, T. K. 1993. A mathematical model of the finding of usability problems, INTERCHI'93, The Netherlands.

[25] Nan, N., Yu, Y., Baixauli, B., Ernst, N., Leite, J., Mylopoulos, J. 2009. Aspects across Software Life Cycle: A Goal-Driven Approach, Transactions on AOSD, Vol. VI, Springer.

[26] Sánchez, P., Moreira, A., Fuentes, L., Araújo, J., Magno, J. 2010. Model-driven development for early aspects. Information & Software Technology, Elsevier, Vol. 52(3).

[27] Whittle, J., Jayaraman, P. 2007. MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation, MoDELS'07, AOM Workshop, USA.

[28] Monteiro, R. 2010. "Model-Driven Requirements Engineering: The Case of Goal-Oriented Approaches", MSc dissertation, FCT, Universidade Nova de Lisboa.

[29] Whittle J., J. Schumann, J. 2000. Generating Statechart Designs from Scenarios, ICSE'00, Limerick, Ireland.

[30] Gutiérrez, J., Nebut, C., Escalona, M., Mejías, M., Ramos, I. 2008. Visualization of Use Cases through Automatically Generated Activity Diagrams, MoDELS'08, Toulouse, France.

[31] Ameller, D., Franch, X., J. Cabot, J. 2010. Dealing with Non-Functional Requirements in Model-Driven Development, RE 2010, Sydney, Australia.

[32] Patrício, P., Amaral, V., Araújo, J., Monteiro, R. 2011. Towards a Unified Goal-Oriented Language, COMPSAC 2011, Germany.