Adding Interoperability to Requirements Models

RUI MONTEIRO, JOÃO ARAÚJO, VASCO AMARAL, MIGUEL GOULÃO, AND PEDRO PATRÍCIO
Universidade Nova de Lisboa, Portugal

Complex software systems inherently require a variety of models used in all of the development stages. A general concern is to guarantee consistency and traceability among these models. Model-driven development (MDD) can help tackle this concern. Although MDD has been mainly used in later development stages, it is relatively unexplored in requirements engineering. In this article, the authors discuss how to leverage MDD to support consistency and traceability in requirements modeling. To illustrate this, they apply MDD to goal-oriented requirements engineering (GORE) by making bidirectional mappings between two well-known GORE approaches (i* and KAOS). The result is an interoperable framework that can be used to migrate from one goal model to another through automatic model transformations, keeping consistency and traceability, so requirements engineers can make the best use of each approach.

Key words

domain specific language, goal-oriented requirements engineering, model-driven development

INTRODUCTION

Complex software systems are characterized not only by a large number of software artifacts (for example, models, components), but also by their intrinsic diversity (Vangheluwe, DeLara, and Masterman 2002). At the modeling level, no single notation is able to capture all of the relevant aspects of those artifacts. Different kinds of models can be used to capture these different perspectives, but the challenge is to make all of these models consistent and to establish traceability among them.

Model-driven development (MDD) has a recognized role in software development in general due to its capabilities of producing consistent and traceable models. The main focus of MDD has been on the design and implementation levels. But requirements engineering (RE) could also benefit from its advantages by allowing different kinds of requirements model transformations between different paradigms, or inside the same paradigm, thus providing support for efficiently capturing, tracing, and handling the different perspectives.

To illustrate and discuss this challenge, the authors apply MDD to goal-oriented requirements engineering (GORE). GORE has a great impact and importance in the RE community (Lamsweerde 2001). It helps in identifying, organizing, and structuring requirements, as well as in exploring and evaluating alternative solutions to a problem (Lamsweerde 2001; Regev and Wegmann 2005). In the context of GORE there are a wide variety of goal-modeling languages, such as i* (Yu 1995),

knowledge acquisition in automated specification (KAOS) (Lamsweerde 2009), goal-based requirements analysis method (GBRAM) (Anton 1996), goal-oriented requirement language (GRL) (ITU-T 2008), and non-functional requirements (NFR) framework (Chung et al. 2000). Despite the existing work on these approaches, consistent and traceable mappings of models from different approaches remain a challenge (Matulevičius and Heymans 2007). For example, it may be desirable to convert one model into another model because one may want to: refine an organizational-level model (for example, using i* models) to a system-level model (for example, by using a KAOS goal model) and vice versa; compare different approaches and decide which one is more expressive to capture a set of requirements (that is, an approach can be more expressive than another to represent certain concepts—for example, obstacles are represented explicitly in KAOS, but not in i*); or facilitate the communication between professionals specialized in different approaches.

The main contribution of this work is to offer an approach to perform the mapping and support traceability between different models. In particular, the authors propose a model transformation framework, called MDGore, to transform i* models into KAOS models and vice versa, through rules defined in the Atlas Transformation Language (Jouault 2008). This article extends (Monteiro et al. 2012) where the problem was tackled solely in mapping i* models to KAOS, completing the approach.

RELATED WORK

Nan et al. (2009) propose a framework 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. This approach uses model-to-code transformation, while the authors' approach uses model-to-model transformation.

A semi-automatic approach, presented in (Sánchez et al. 2010), 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. One-way transformations are made from requirements to architecture. In contrast, the authors' approach makes bidirectional transformations at the requirements level.

A UML-based modeling tool, called MATA, uses graph transformations to specify and compose aspects (Whittle and Jayaraman 2008). Its main goal is to compose aspects in UML models. In contrast with the authors' approach, MATA graph rules are defined over the concrete syntax of the modeling language, in this case the UML. Comparing this approach with the authors' approach, MATA focuses on a particular kind of transformation, that is, composition.

In Ameller, Franch, and Cabot (2010), the current state of MDD approaches with respect to NFRs is reported. In general, NFRs are not addressed in MDD methods and processes. A general framework that integrates NFRs into the core of the MDD process is outlined, but no full-fledged approach is proposed.

In (Patrício et al. 2011), the authors propose a new extensible language to unify and represent GORE languages. The unifying treatment of concepts in their model makes it possible to define more projections of each problem than the existing languages by themselves. Also, hybrid models can be used. However, if it is not possible or desirable to adopt a unifying language, a transformation-based approach, proposed in this article, is a plausible alternative.

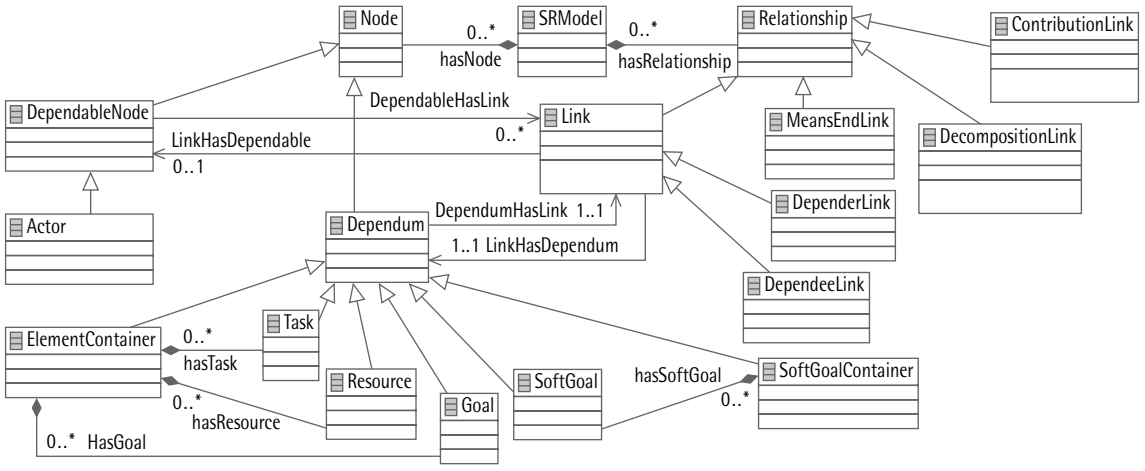
BACKGROUND

This section introduces the MDD and GORE. The authors chose KAOS and i* to discuss and illustrate the transformations, as they are among the most popular GORE approaches.

Model-Driven Development

MDD (Völter and Stahl 2006) is a paradigm that promotes 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. In MDD, a model is defined by a metamodel. In other words, a model must be constructed following certain rules and conform to a certain metamodel. In the same way, the metamodel is a model of a modeling language and must conform to the rules defined in the meta-metamodel (for example, MOF). 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

FIGURE 1 Partial LDE i* metamodel



takes as input a model conforming to a given metamodel and produces as output another model in conformance to a given target language metamodel.

Model transformation languages aim at automating the process of deriving one model from another one. The authors used ATL in their project for specifying their model transformations, as it is very well documented and supported.

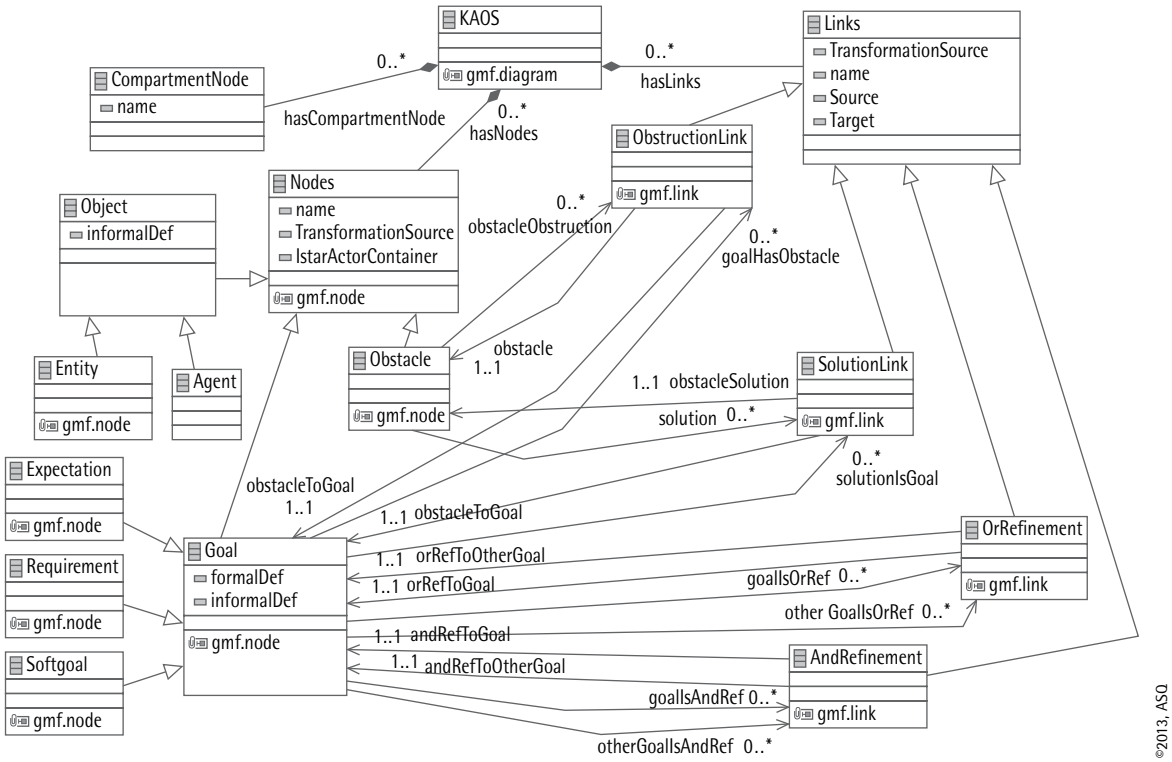
The i* Approach

i* was developed for modeling and reasoning about organizational environments and their information systems. It focuses on the concept of intentional actor. Actors in their organizational environment have 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 (depender) depends on another actor (dependee) to achieve goals and softgoals (for example, quality attributes), 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) to specify alternative ways to achieve goals. Decomposition links are used to decompose tasks. A task can be decomposed into subgoal, subtask, resource, and softgoal. Also, there are positive and negative contribution links.

The authors use LDE i* (Nunes et al. 2009), a domain-specific language (DSL) (Kelly and Tolvanen 2008), for i*. Its main objective is to ensure a better management of complexity and scalability of i* models by introducing the notion of a compartment where elements of an i* model can be grouped. This DSL was implemented using MDD techniques, and includes an Ecore (Steinberg et al. 2008) metamodel based on existing ones (Yu 1995; Alencar et al. 2008). Figure 1 shows a fragment of the LDE i* metamodel. The full version can be found in (Nunes et al. 2009).

The root of the metamodel is the class SR model. 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, specialized as an actor, or a dependum. 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 tasks, goals, and resources. The softgoalcontainer compartment groups 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 eclipse plug-ins (Steinberg et al. 2008; Richley 2007).

FIGURE 2 Partial modularKAOS metamodel



The KAOS Approach

KAOS is a systematic approach for discovering and structuring system-level requirements. 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. 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" as a situation that prevents the achievement of a goal. Usually the solution to the obstacle is expressed as a requirement.

The authors use the modularKAOS DSL (Dias, Amaral, and Araújo 2009) for dealing with KAOS models. ModularKAOS incorporates the notion of compartment to handle model complexity. This DSL was implemented based on a metamodel defined in (Matulevičius and Heymans 2007), using Ecore. Figure 2 shows a partial modularKAOS metamodel. The full version can be found in Dias, Amaral, and Araújo (2009).

The class KAOS is the metamodel root. This is composed of zero or more compartmentnodes, nodes, and links. The nodes class can be object, goal, or obstacle. An object can be agent or entity. A goal can be an expectation, requirement, or softgoal. Links can be: obstructionlink, solutionlink, OrRefinement, or AndRefinement. Obstructionlink connects an obstacle to a goal. Solutionlink connects a goal to an obstacle. Links OrRefinement and AndRefinement refine goals.

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 from i* SR into KAOS goal models and vice versa. Figure 3 gives an overview of the authors' framework. The dashed rectangle is the focus of their approach. If the source model is the i* SR model, this is transformed into a KAOS goal model through the application of rules implemented in ATL. Likewise, if the source

is the KAOS goal model, this is transformed into an i* SR model.

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 Ecore 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 and vice versa. These specific cases will be addressed in the next section. The authors defined a metamodel, illustrated in Figure 4, to generate a "decision model" so the users can make and record their choices.

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 the authors 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 nontransformed elements of the i* SR model.

FIGURE 3 Process overview of the MDGore framework

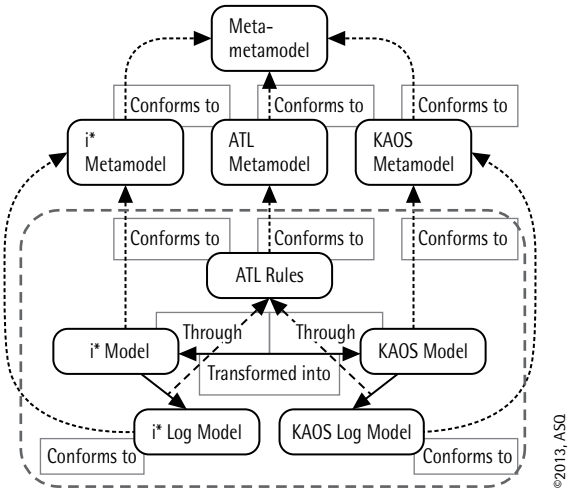
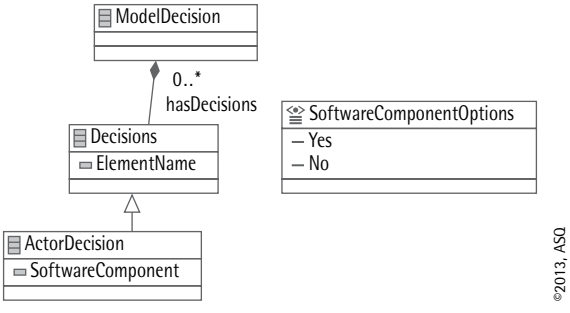


FIGURE 4 Metamodel of the decision model



Another concern the authors had was to ensure traceability in the transformation process. Through attributes in the KAOS metamodel, they 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, the authors' 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.

TABLE 1 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
MeansEndLink	AndRefinement; OrRefinement
Is-part-of Association	Aggregation
ISA Association	Inheritance
Break, Some-, Hurt	Conflict
Make, Some+, Help	OrRefinement
Dependency	AndRefinement

Mapping Between i* and KAOS Elements

The relations between i* and KAOS elements have been established based on the concepts of the two approaches introduced previously and on the abstract syntax of each approach. Table 1 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, while 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. 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 (omitted in Figure 2).

If a task is decomposed into subtasks, 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 MeansEndlink has two possible mappings: AndRefinement or OrRefinement. If the source element of the MeansEndLink has another MeansEndLink, each of the destination elements is seen as an alternative to obtain the same source element. In this case, each MeansEndLink is mapped to an OrRefinement. On the other hand, if the source element has only one MeansEndLink, 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 intervention is needed.

Example of Transformation

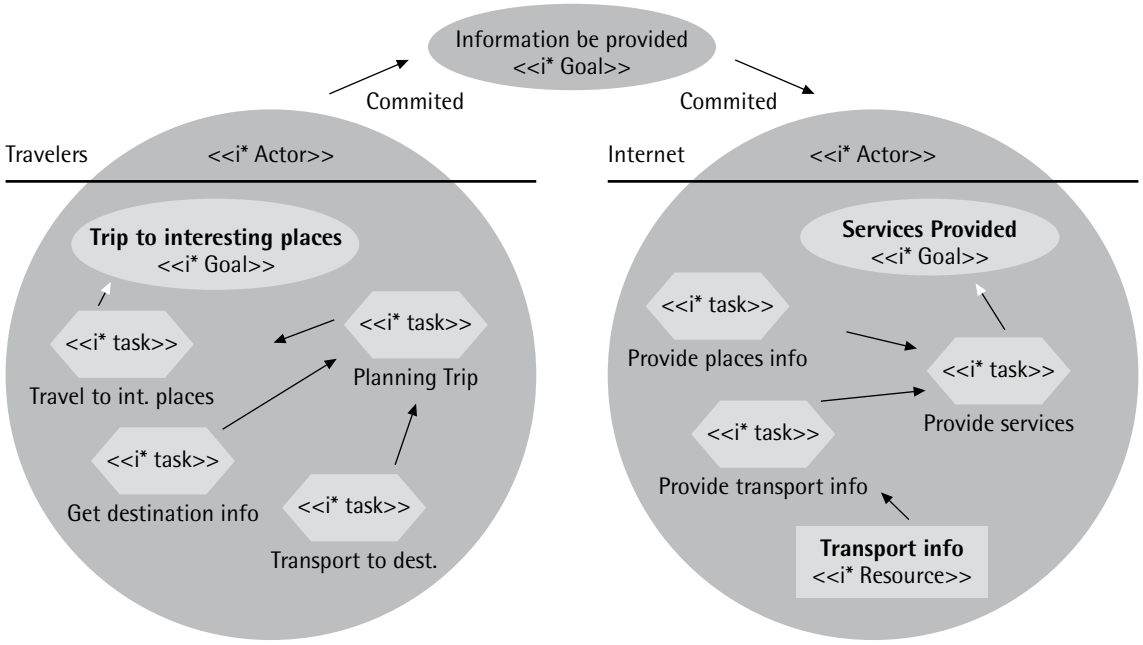
To help explain the transformation process, the authors introduce a partial i* SR model (illustrated in Figure 5) of the "Project BTW: if you go, my advice to you" (Lucena et al. 2009). The main goal of this project is to develop a route planning system that allows community input. The model contains 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."

Transformation Process

In this section the authors describe the transformation process of an i* SR model into a KAOS goal model. This transformation process is based on the mappings

FIGURE 5 Partial i* SR model of the BTW project



described previously, and consists of: 1) generation of a decision model; 2) generation of an annotated i* SR model; 3) generation of an intermediate KAOS goal model; 4) generation of the final KAOS goal model; and 5) generation of a log model. For each step, a set of ATL transformation rules was specified and implemented.

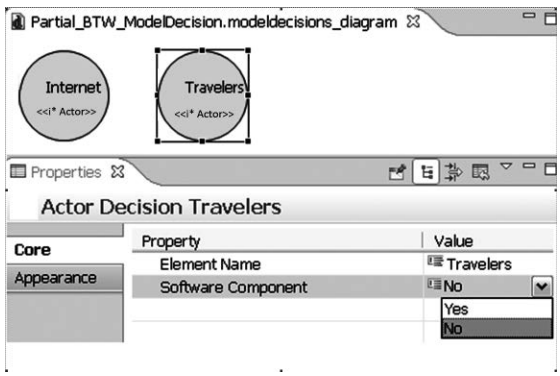
Step 1: Generation of a decision model

In this step a decision model from the i* SR model is generated, so 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 the actor should be mapped to. Figure 6 presents a decision model 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 the actor is part of the system, and in this case it will then be mapped to a SystemAgent. The "no" option means the actor is an intervening one in the system and will then be mapped to an EnvironmentAgent.

In the example for the travelers actor, the "no" option in the property SoftwareComponent was

FIGURE 6 Decision model example



selected, which means it will be mapped into an EnvironmentAgent. Although Figure 6 does not show it, for the Internet actor the "yes" option was selected, which means it will be mapped to a SystemAgent.

Step 2: Generation of an annotated i* SR model

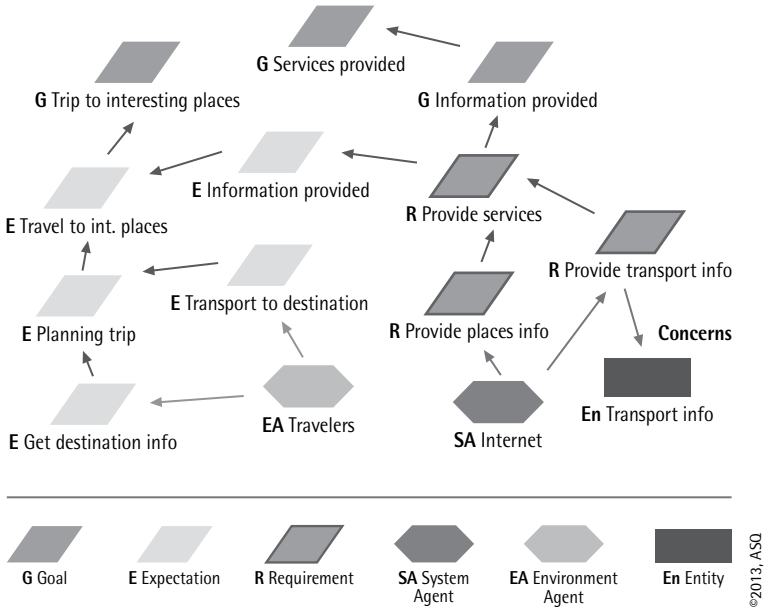
After taking all of the decisions in the previous step, this step generates an annotated i* SR model, from the decision model and the original i* SR model, with all of these decisions. In fact, the annotated model

contains the same elements as the original i* SR model, and is enriched with user decisions. From this step on, all of 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 i* SR model built in 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 being a task. However, this solution does not fully solve the problem, since the responsibility links between agents and requirements or expectations are not automatically derived. This problem is resolved in the next step.

FIGURE 7 KAOS goal model obtained



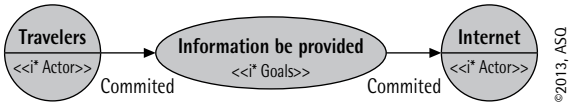
Step 4: Generation of the final KAOS goal model

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, an ATL rule was created to restore the responsibility links between the elements and their agents. These responsibility links are visible in 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 of this step is to generate a log model from the annotated i* SR model in the second step, with all of the i* elements that are impossible to transform. Note that this log model will also include model elements that were transformed, but were related to elements that could not be transformed, so their relationships are preserved through the log model. For example, in the i* SR model of Figure 5, the dependency relationship between actors is impossible to transform. This relationship is stored in the log model, shown in Figure 8, together with the actors (travelers and Internet) and the goal (information be provided) in the center of the dependency. Although these elements (actors and goal) have been transformed, they appear in the log model because they are part of the dependency link that was not transformed, because it is not possible to store such a link without its source and target elements. If all of the SR model elements are transformable, an empty log model is generated. Otherwise, a log model is generated with the elements that are not transformed.

FIGURE 8 Log model



Transformation Result

The output of the transformation process explained previously is a KAOS goal model with the transformed

elements and a log model with the i* elements that cannot be transformed. In the KAOS goal model, illustrated in Figure 7, one can see that the i* goals were transformed into KAOS goals. The travelers actor was transformed into an EnvironmentAgent, as in the decision model; the authors decided that it is not a software component. For the Internet actor in the same decision model, the authors 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. Similarly, the internal tasks of the Internet actor were transformed into requirements, since this actor corresponds to a SystemAgent. The internal resources to the Internet actor were transformed into entities.

All decomposition links between tasks were transformed into AndRefinement links and decomposition links between tasks and resources were transformed into ConcernsLinks. The MeansEndLinks between the goals and tasks were transformed into AndRefinement, as each goal has only one MeansEndLink.

The authors' 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 (see Figure 8).

The authors changed the KAOS goal model to establish a possible solution to this dependency relationship. The solution was to represent the dependum "information be provided" for both perspectives, that is, travelers' and Internet's. First, they specified the expectation "information be provided" (traveler's perspective) resulting from the refinement of the expectation "travel to interesting places." Second, 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. The authors also created the goal "information be 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.

TABLE 2 Mappings of KAOS elements into i* elements

KAOS	i*
Goal	Goal (external)
	Goal (internal)
Requirement	Task (external)
	Task (internal)
	Goal (internal)
Expectation	Task (external)
	Task (internal)
	Goal (internal)
Softgoal	Softgoal (external)
	Softgoal (internal)
Operation	Task (external)
	Task (internal)
Obstacle	Softgoal (external)
	Softgoal (internal)
Domain properties	Softgoal (external)
	Softgoal (internal)
Entity	Resource (external)
	Resource (internal)
System Agent, Environment Agent	Position
	Role
	Agent
	Actor
OrRefinement	Means–End link
	Help
	Some+
	Make
	Decomposition
AndRefinement	Decomposition
	Means–End link
	Dependency
	Make
ObstacleLink	Break
ObstacleRefinement	Make
Solution	Break
Concerns	Decomposition
DomainPropertiesLink	Decomposition
OperationalizationLink	Decomposition
Inheritance	ISALink
Aggregation	IsPartOfLink
Conflict	Break
	Some–
	Hurt

Mapping From KAOS to i* Elements

Table 2 shows the mappings that must be carried out from KAOS to i*. This mapping direction brings different possibilities for each model element, requiring more interventions from the stakeholder to decide the most appropriate target model element.

Some KAOS model elements are mapped into i* model elements that can be external or internal to the i* actor. A model element is external if it is not linked to any element that is under the responsibility of the agent, and internal otherwise. For example, a KAOS goal is mapped to an i* goal, but this can be inside or outside an i* actor, that is, internal or external to the actors.

Requirements and expectations can be mapped either to internal goals (as they are always directly linked to an agent) or internal/external tasks. Similarly, softgoals are mapped to internal or external softgoals. Operations can also be mapped to either internal or external tasks.

Concerning obstacles, there is no direct correspondence. The closest one is to softgoals. The same applies to domain properties. Entities can be mapped to internal or external resources. System and environment agents can be mapped to actor, agent, position, or role.

Regarding relationships, OrRefinements can be mapped into a means-end link, contribution links, or an alternative decomposition link. AndRefinements are mapped into and decomposition, make contribution, a dependency, or a means-end link. Obstacle links, refinements, and resolutions can be mapped into contribution links: break, make and break, respectively. Concerns, domain properties, and operationalization links are mapped into decomposition links. Inheritance and aggregation of objects are mapped into isALink and a IsPartOfLink. Finally, conflict links can be mapped into the contribution links.

By transforming the KAOS model in Figure 7, the authors obtain the i* SR model in Figure 5. Due to space constraints, they do not present the KAOS log model, which is considerably larger than the log model for the transformation from i* to KAOS.

Validation

The authors assessed their transformation rules' correctness by applying the approach to the Health Watcher case study, which is a real-world system that provides information about public health (Massoni, Soares, and Barba 2007). The detailed model of the Health Watcher system built with the authors' approach can be found in (Soares, Barba, and Laureano 2006).

The authors also conducted a pilot study with seven final-year master's degree students in computer science at the Universidade Nova de Lisboa, who had previous experience with GORE and MDD, to assess the effect of the proposed approach and its tool support in GORE activities. A group of five testers is normally enough to uncover more than 80 percent of the usability problems (Nielsen and Landauer 1993). 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 of training. After completing their training, they were asked to perform a set of modeling tasks. First, they manually created a KAOS goal model based on the i* SR model of the BTW project. Then, they transformed the i* SR model into the KAOS goal model and compared it with the goal model they created manually. A similar process was followed for the transformation from KAOS to i*. Upon completion of these tasks, participants were asked to fill in a questionnaire to compare the approach with the baseline manual approach.

The questionnaire included four questions, similar for both transformation processes:

- Q1: "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)
- Q2: "Was there any information lost in the transformation process?" (1—yes; 5—no)
- Q3: "How do you rate the simplicity of the transformation process?" (1—complex; 2—little simple; 3—reasonably simple; 4—simple; 5—very simple)
- Q4: "How do you rate the usefulness of the approach?" (1—useless; 2—little useful; 3—somehow useful; 4—useful; 5—very useful)

Table 3 summarizes the responses, broken down by question, for both transformation processes denoted by the label (i*-k) when transforming from i* to KAOS and (k-i*), when transforming from KAOS to

i*. Note that there are only two categories for question 2, while there are five for the remaining questions. The authors aggregate the answers from categories 1, 2, and 3 as no improvement, denoted by "☹/😐" and those with categories 4 and 5 as improvement, denoted by "☺." Finally, the authors present the chi-square test for the aggregated answers.

TABLE 3 Questonnaire's answers frequency, by question

	Responses					Aggregate		Chi-Square Test (Aggr.)		
Questions	1	2	3	4	5	☹/😐	☺	Chi-sqr	df	Sig
Q1 (i*-k)	0	0	0	2	5	0	7	7.000	1	0.008
Q2 (i*-k)	0				7	0	7	7.000	1	0.008
Q3 (i*-k)	0	1	5	1	0	6	1	3.571	1	0.059
Q4 (i*-k)	0	0	0	1	6	0	7	7.000	1	0.008
Q1 (k-i*)	0	0	0	3	4	0	7	7.000	1	0.008
Q2 (k-i*)	0				7	0	7	7.000	1	0.008
Q3 (k-i*)	0	1	6	0	0	7	0	7.000	1	0.008
Q4 (k-i*)	0	0	0	1	6	0	7	7.000	1	0.008

The aggregate results are consistent for both transformations, and show improvements on questions 1, 2, and 4, with statistical significance, although the authors must stress that with only seven participants, it would be advisable to conduct replications of this study. All participants rated favorably the generated models and the usefulness of the approach. None of the participants reported information losses in the process. They particularly enjoyed the support for interoperability among teams with different expertise in GORE, which was one of the planned contributions of this approach.

The answers to question 3 were mostly neutral (the transformation process was considered "reasonably simple"). 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 for setting up the transformation process in the tool support would mitigate this problem.

CONCLUSIONS

MDD can be used successfully to relate models of different approaches, in different abstraction levels, that share the same paradigm. The MDGore approach brings the following contributions to the requirements community:

- A bidirectional, traceable mapping with no information loss, thanks to the use of log models
- Decision support concerning which approach is more expressive than another in a given context
- Facilitated communication between professionals specialized in different approaches

In this article, the authors expanded a previous work (Monteiro et al. 2012) by producing a complete semi-automatic model-to-model transformation framework, to relate KAOS and i* models in both directions. They validated their approach by applying it to two case studies and via a questionnaire. This preliminary evidence confirms the framework's usefulness.

The authors' future work includes developing transformations to other requirements approaches using different paradigms.

ACKNOWLEDGMENTS

The authors would like to acknowledge CITI—PEst—OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL)—2011-2012)—for the financial support for this work.

REFERENCES

Alencar, F., C. Silva, M. Lucena, J. Castro, E. Santos, and R. Ramos. 2008. Improving the understandability of i* models. In *Proceedings of the 10th International Conference on Enterprise Information Systems* (ICEIS'08), June 12-16, Barcelona, Spain: SAIC.

Ameller, D., X. Franch, and J. Cabot. 2010. Dealing with non-functional requirements in model-driven development. In *Proceedings of the 18th IEEE International Requirements Engineering Conference* (RE'10), September 27-October 1. Sydney, Australia: IEEE Computer Society.

Antón, A. 1996. Goal-based requirements analysis. In *Proceedings of the Second IEEE International Conference on Requirements Engineering (ICRE'96),* April 15-18. Colorado Springs, CO: IEEE Computer Society.

Chung, L., B. Nixon, E. Yu, and J. Mylopoulos. 2000. *NFR in software engineering.* Netherlands: Kluwer.

Czarnecki, K., and S. Helsen. 2003. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the context of MDA,* October 27, Anaheim, CA.

Dias, A., V. Amaral, and J. Araújo. 2009. Towards a domain specific language for a goal-oriented approach based on KAOS. In *Proceedings of the Third IEEE International Conference on Research Challenges in Computer Science (RCIS'09),* April 22-24. Fès, Morocco: IEEE Computer Society.

ITU-T. 2008. Recommendation Z.151 (09/08). User Requirements Notation (URN)—Language definition. Geneva, Switzerland.

Jouault, F., F. Allilaire, J. Bézivin, and I. Kurtev. 2008. ATL: A model transformation tool. *Science of Computer Programming,* vol. 72 no.1: Elsevier.

Kelly, S., and J. Tolvanen. 2008. *Domain specific modeling.* New York: Wiley.

Lamsweerde, A. 2001. *Goal-oriented requirements engineering: A guided tour. In Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01),* August 27-31. Toronto, Canada: IEEE Computer Society.

Lamsweerde, A. 2009. *Requirements engineering: From system goals to UML models to software specifications.* New York: Wiley.

Lucena, M., J. Castro, C. Silva, F. Alencar, E. Santos, and J. Pimentel. 2009. A model transformation approach to derive architectural models from goal-oriented requirements models. In *Proceedings of the On The Move to Meaningful Internet Systems: OTM 2009 Workshops,* November 1-6. Vilamoura, Portugal: Springer.

Massoni, T., S. Soares, and P. Borba. 2007. Requirements health-watcher version 2.0. In *Proceedings of the Early Aspects at ICSE: Workshop in Aspect-Oriented Requirements Engineering and Architecture Design,* May 20-26, Minneapolis, MN.

Matulevičius, R., and P. Heymans. 2005. Analysis of KAOS meta-model. Technical Report, University of Namur, Belgium.

Matulevičius, R., and P. Heymans. 2007. Comparing goal modelling languages: An Experiment. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'07),* June 11-12. Trondheim, Norway: LNCS 4542, Springer-Verlag.

Monteiro, R., J. Araújo, V. Amaral, M. Goulão, and P. Patrício. 2012. Model-driven development for requirements engineering: The case of goal-oriented approaches. In *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012),* September 3-6. Lisbon, Portugal: IEEE CPS.

Monteiro, R., J. Araújo, V. Amaral, and P. Patrício. 2010. MDGore: Towards model-driven and goal-oriented requirements engineering. In *Proceedings of the 18th International Conference on Requirements Engineering (RE'10),* September 27-October 1. Sydney, Australia: IEEE Computer Society.

Nan, N., Y. Yu, B. Baixauli, N. Ernst, J. Leite, and J. Mylopoulos. 2009. Aspects across software life cycle: A goal-driven approach. *Transactions on AOSD,* vol. VI.

Nielsen, J., and T. K. Landauer. 1993. A mathematical model of the finding of usability problems. In *Proceedings of INTERCHI'93,* April 24-29. Amsterdam, Netherlands: ACM.

Nunes, C., J. Araújo, V. Amaral, and C. Silva. 2009. A domain specific language for the i* framework. In *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS'09),* May 6-10. Milan, Italy: LNBIP 24, Springer.

Patrício, P., V. Amaral, J. Araújo, and R. Monteiro. 2011. Towards a unified goal-oriented language. In *Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2011),* July 18-22. Munich, Germany: IEEE Computer Society.

Regev, G., and A. Wegmann. 2005. Where do goals come from: The underlying principles of goal-oriented requirements engineering. In *Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05),* August 29-September 2. Paris, France: IEEE Computer Society.

Richley, J. 2007. GMF: *Beyond the wizzards.* Available at: http://www.onjava.com/.

Sánchez, P., A. Moreira, L. Fuentes, J. Araújo, and J. Magno. 2010. Model-driven development for early aspects. *Information & Software Technology* vol. 52, no. 3: Elsevier.

Soares, S., P. Borba, and E. Laureano. 2006. Distribution and persistence as aspects. *Software: Practice and experience* vol. 36, no. 7.

Steinberg, D., F. Budinsky, M. Paternostro, and E. Merks. 2008. *EMF eclipse modeling framework.* Reading, MA: Addison-Wesley.

Vangheluwe, H., J. De Lara, and P. J. Mosterman. 2002. An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 Conference (AI, Simulation and Planning in High Autonomy Systems),* April 7-10, Lisbon, Portugal.

Völter, M., and T. Stahl. 2006. *Model-driven software development—Technology, engineering, management.* New York: Wiley.

Whittle, J., and P. Jayaraman. 2008. MATA: A tool for aspect-oriented modeling based on graph transformation. In *Models in Software Engineering.* Berlin: Springer-Verlag.

Yu, E. 1995. Modelling strategic relationships for process reengineering. Ph.D. thesis, University of Toronto, Canada.

BIOGRAPHIES

Rui Monteiro holds bachelor's and master's degrees in informatics from FCT/UNL, Portugal. He currently works as a software engineering consultant at Aubay and BNP Paribas Net. He can be reached by email at rm.chambel@gmail.com.

João Araújo holds a doctorate from Lancaster University, U.K., and is an assistant professor at FCT/UNL, Portugal. His principal research interests are in requirements engineering, model-driven engineering, and software product lines where he has published circa 200 articles on journals and conferences. He was a co-founder of the Early Aspects and MoDRE workshops and a contributor to several European and national research projects. He has served on the program and organizing committees for top conferences such as RE, MoDELS, CAiSE, ICSE, ER, and AOSD. He can be reached by email at joao.araujo@fct.unl.pt.

Vasco Amaral is assistant professor at FCT/UNL and full member of CITI. He holds a doctorate from the University of Mannheim, Germany, and worked in the past as software engineer on high-energy physics computing and very large databases at CERN (Switzerland), DESY (Germany), and LIP (Portugal). Amaral's research interests include software languages engineering, model-driven development, verification, model composition and transformations, and multiparadigm modeling. He serves on the program and organizing committees of several international conferences and workshops and is the main organizer of the domain-specific modeling theory and practice summer school series. He can be reached by email at vma@fct.unl.pt.

Miguel Goulão earned his doctorate in informatics from FCT/UNL (Portugal) in 2008. He is assistant professor of the Informatics Department of FCT/UNL, a member of the CITI research group, and one of the founding members of the Portuguese Association for Quality in Information and Communication Technologies. His research interests include experimental software engineering, and how it can be applied to support quality improvement throughout the software process, from requirements elicitation to software evolution. He has published more than 45 peer-reviewed papers and served as a reviewer in several journals, conferences, and workshops. He can be reached by email at mgoul@fct.unl.pt.

Pedro Patrício holds bachelor's and master's degrees in informatics from FCT/UNL, Portugal. He currently works as a software engineering consultant at Link. He can be reached by email at pedropt@gmail.com.