A Change Impact Analysis Approach to GRL Models

Jameleddine Hassine
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
Email: jhassine@kfupm.edu.sa

***Abstract*—Goal models represent interests, intentions, and strategies of different stakeholders in early requirements elicitation process. In a socio-technical context, goal models evolve quickly to accommodate the rapid changes of stakeholders' needs, technologies, and business environments. In order to control and minimize the risks brought by requirements changes, it is important to analyze the effects of modifications in goal models. Given a proposed modification, Change Impact Analysis (CIA) allows for the identification of software artifacts that will be impacted and for the estimation of the effort required to implement the proposed changes. This paper describes an approach to analyze the propagation of changes in goal models specified using the Goal-oriented Requirements Language (GRL). Dependencies in the GRL model are first extracted and illustrated using a GRL Model Dependency Graph (GMDG), describing inter- and intra- actor dependencies. In order to identify model constructs that are impacted by a proposed change, we apply the well-known technique of program slicing to the GMDG model. We illustrate our approach by applying it to a goal model describing undergraduate students' involvement in research activities.**

***Keywords-Goal models*; *requirements*; *Change Impact Analysis*; *Goal-oriented language(GRL)*; *GRL Model Dependency Graph (GMDG)*; *slicing*;**

I. INTRODUCTION

Evolving customer needs is one of the driving factors in software development. Requirements models are the first available artifacts during the software development process. They undergo many changes caused by changing user requirements and business goals or induced by changes in implementation technologies. Hence, there is a need to analyze the impact of requirements changes in order to help detect and solve possible conflicts between stakeholders and to assess the different design alternatives influenced by these changes. Localizing the impact of changes is one of the most efficient strategies for a successful evolution. Change Impact Analysis (CIA) [1] aims at identifying the potential consequences of a change, and estimating what needs to be modified to accomplish a change [1]. Hence, change impact analysis is required for constantly evolving systems to support the comprehension, implementation, and evolution of changes. Change impact analysis has been applied to analyze source code, formal models (architectural and requirements models), or other artifacts (e.g., documents, data sources, configuration files) [2].

Goal models have been introduced as a means to ensure that stakeholders' interests and expectations are met in the early requirements engineering stages. As goal models gain in complexity (e.g., large systems involving many stakeholders and many dependencies), they become difficult to comprehend and to maintain. The main motivation of this research is to manage the complexity of goal models with respect to maintenance tasks. In particular, we are interested in applying change impact analysis techniques to Goal-oriented Requirement Language (GRL) [3]. This paper aims to:

- Provide a GRL-based approach to change impact analysis. The proposed approach allows requirements engineers and projects leaders to ask "what if ...?" questions, and to reason about alternative scenarios with respect to maintain GRL models.
- Provide an insight into how changes propagate within an actor or how they spread across many actors. In fact, dependencies that are embedded within the GRL model are extracted and described as a GRL Model Dependency Graph (GMDG). Dependencies between intentional elements that are within one GRL actor are referred to as intra-actor dependencies, while dependencies involving different actors are referred to as inter-actor dependencies. In case of a modification of the GRL model, it is important to identify whether the changes strech across the actor boundary. This would allow to choosing the correct validation approach (e.g., mediation process or a less formal discussions between the intervening stakeholders).
- Extend the use of the well-known technique of program slicing [4] to goal models. In contrast to traditional program slicing, we apply slicing to GMDG graphs. As a result, the GRL artifacts that are impacted by a given change are identified.

The remainder of this paper is organized as follows. In the following section, we provide a brief introduction to the GRL [3] language. Our proposed change impact analysis approach is presented in Section III. In Section IV, we apply our proposed approach to a GRL model describing the involvement of undergraduate students and professors in research activities. A comparison with related work is provided in Section V. Finally, conclusions and future work are presented in Section VI.

II. GOAL-ORIENTED REQUIREMENTS LANGUAGE (GRL)

GRL [3] is a visual notation used to model stakeholders' intentions, business goals, and non-functional requirements (NFR).

The basic notational elements of GRL are summarized in Figure 1. Actors (see Figure 1(a)) are holders of intentions; they are the active entities in the system or its environment

 ISBN: 978-1-61208-449-7

who want goals to be achieved, tasks to be performed, resources to be available, and softgoals to be satisfied. Actor definitions are often used to represent stakeholders as well as systems. A GRL actor may contain intentional elements describing its intentions and capabilities. Figure 1(b) illustrates the GRL intentional elements (i.e., goal, task, softgoal, resource and belief) that optionally reside within an actor.

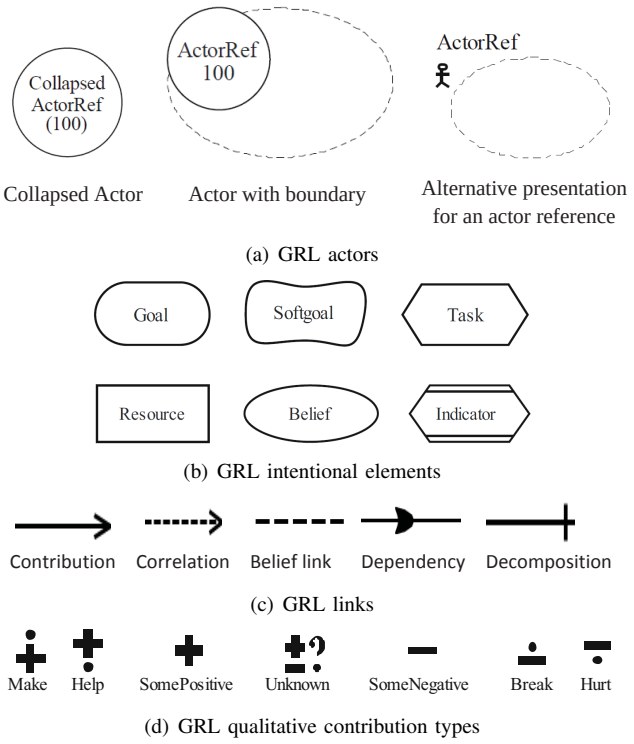


Fig. 1. GRL notational elements

Figure 1(c) illustrates the various kinds of links in a goal model. Decomposition links allow an intentional element to be decomposed into sub-elements (using AND, OR, or XOR). Beliefs, used to represent rationales from model creators, are connected to GRL intentional elements using belief links. Contribution links indicate desired impacts of one intentional element on another element. A contribution link has a qualitative contribution type (see Figure 1(d)) and/or a quantitative contribution (an integer value between -100 and 100 in standard GRL). Correlation links are similar to contribution links but describe side effects rather than desired impacts. Finally, dependency links model relationships between actors and between intentional elements (within the same actor). In addition, GRL defines *indicators* as containable elements used to analyze GRL models based on real-world measurements. Since, indicators are used only in converting real-world values into satisfaction levels, they are out of the scope of this research. For a detailed description of the GRL language, the reader is invited to consult the URN standard [3].

III. GRL Change Impact Analysis Approach

In what follows, we present our GRL-based change impact analysis approach.

A. Dependencies in GRL Models

Goal-oriented models describe the actors within a complex socio-technical system, dependencies between system elements, and organizational goals. Dependencies enable reasoning about how actors/elements depend on each other to achieve the planned goals. Dependencies in GRL models can be classified as explicit or implicit [5]. Explicit dependencies are modeled as dependency links, while implicit dependencies are modeled using contributions, correlations, and decompositions [5]. For instance, in Figure 2(a), the satisfaction of *Goal1* depends on the satisfaction of *Goal2* and the type of the contribution *ContributionG2G1*. A change to the satisfaction level of *Goal2* or to the contribution's qualitative/quantitative type/level, will have a direct impact on the satisfaction of *Goal1*. Explicit dependency links can be used in many types of configurations according to the required level of detail [3]. GRL actors (described as collapsed actors) can be used as source and/or destination of an explicit dependency link. Intentional elements inside actor definitions can be used as source and/or destination of a dependency link. Collapsed actors (see Figure 1(a)) are used when an actor is the source/destination of an explicit dependency. In addition, actors cannot be used as source or destination of a contribution, a correlation, or a decomposition link. We further classify dependencies as:

- *Intra-actor* dependencies: describes a dependency (explicit or implicit) having its source and target within the same actor boundary.
- *Inter-actor* dependencies: describes a dependency (explicit or implicit) having its source and target bound to different actor definitions.

It is worth noting that GRL syntax does not allow actors to overlap (i.e., share common GRL elements).

B. GRL Model Dependency Graph (GMDG)

Before we define the GMDG graph, we define formally GRL models.

Definition 1 (GRL Model): We assume that a GRL model *GRLM* is denoted by a 3-tuple (or a triple): (Actors, Elements, Links), where:

- *Actors* is the set of actors in the GRL model. It contains a set of actor references and a set of collapsed actors, denoted by *CollapsedActors*.
- *Elements* is the set of intentional elements (i.e., tasks, goals, softgoals, resources, and beliefs) in the GRL model.
- *Links* is the set of links in the model. A GRL link is defined as a triple (*type*, *src*, *dest*), where *type* is the link type (of type *LinkTypes* = {contribution, correlation,

dependency, decomposition}), *src* and *dest* are the source and destination of the link, respectively (they are either part of *Elements* or *CollapsedActors*).

Definition 2 (Links Access Functions): We define the following access functions over the set of links (i.e., *Links*):

- *TypeLink*: *Links* → *LinkTypes*, returns the type of the link.
- *Source*: *Links* → *Elements* ∪ *CollapsedActors*, returns the intentional element/actor source of the link.
- *Destination*: *Links* → *Elements* ∪ *CollapsedActors*, returns the intentional element/actor destination of the link.

For example, given a contribution *C*=(*contribution*, *Goal2*, *Goal1*) (see Figure 2(a)), then *TypeLink*(*C*)= *contribution*, *Source*(*C*)= *Goal2*, and *Destination*(*C*)= *Goal1*.

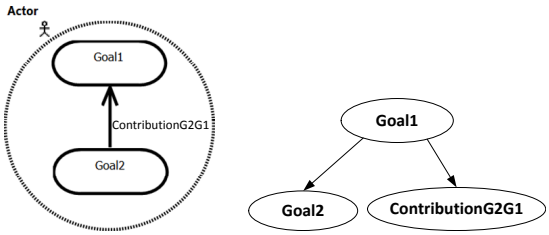
Using the dependency definitions given above, we can now define the GRL Model Dependency Graph (GMDG).

Definition 3 (GRL Model Dependency Graph (GMDG)): A GRL Model Dependency Graph (GMDG) is a directed, connected graph defined as GMDG=(N, E), consisting of:

- *N* is a set of nodes. Each GRL intentional element (i.e., of type *Elements*), each link (i.e., of type *LinkTypes*), and each collapsed actor is mapped to a node *n*∈*N*.
- *E* is a set of edges. An edge *e*∈*E* represents a dependency link (intra- or inter- actor dependency).

C. Constructing the GMDG Graph

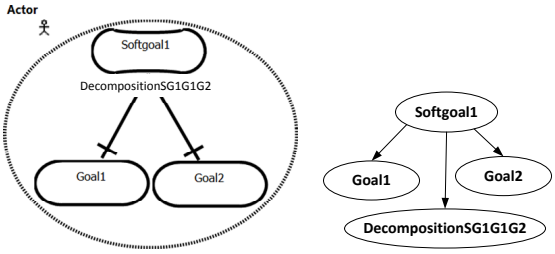
In a GMDG graph, intra-actor dependencies are illustrated as solid arrows (see Figure 2(b)), while inter-actor dependencies are illustrated as dashed arrows (see Figure 4(b)). Figure 2(b) illustrates the GMDG corresponding to a single contribution link (Figure 2(a)). GMDG nodes are created for *Goal1*, *Goal2*, and the contribution link *ContributionG2G1*. Two intra-actor dependency links are created from node *Goal1* to both *Goal2* and *ContributionG2G1* nodes.



(a) Contribution link (b) GMDG for the contribution link

Fig. 2. Contribution relationship and its Associated GMDG

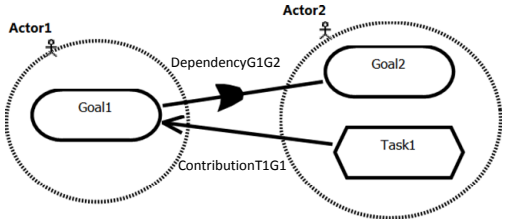
In case of a decomposition relationship (see Figure 3(a)), each intentional element is mapped to a GMDG node (i.e., *Softgoal1*, *Goal1*, *Goal2*). The decomposition is mapped to one single node (i.e., *DecompositionSG1G1G2*). Intra-actor dependencies are created between *Softgoal1* and *Goal1*, *Goal2*, and *DecompositionSG1G1G2* (see Figure 3(b)).



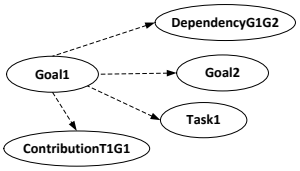
(a) Decomposition Relationship (b) GMDG for the Decomposition Relationship

Fig. 3. Decomposition relationship and its associated GMDG

Figure 4(a) illustrates a GRL having two actors (i.e., *Actor1* and *Actor2*) and two inter-actor dependencies (i.e., *DependencyG1G2* and *ContributionT1G1*). Explicit dependencies are mapped to GMDG nodes as follows: nodes are created to map the depender (e.g., *Goal1*), the dependee (e.g., *Goal2*), and the dependency link (e.g., *DependencyG1G2*). Two inter-actor dependency links are created from node *Goal1* to *Goal2* and *DependencyG1G2* nodes. The contribution link is mapped to two inter-actor dependencies, one between *Goal1* and *Task1*, and the other between *Goal1* and ContributionT1G1 as shown in Figure 4(b).



(a) Inter-actor Dependencies



(b) GMDG with Inter-actor Dependencies

Fig. 4. Inter-actor dependencies and their associated GMDG

It is worth noting that beliefs are always associated with one single intentional element. Consequently, we don't create a separate GMDG node for representing a belief. The algorithm in Figure 5 is used to construct the GMDG graph.

D. Slicing the GRL Model Dependency Graph

The notion of program slicing was originated in the seminal paper by Weiser [4]. Weiser defined a slice *S* as a reduced independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior. Informally, a static program slice consists of those parts of a program that potentially could affect/affected by

```
Input  : A GRL Model
Output: A GMDG Graph
foreach element e ∈ Elements ∪ CollapsedActors do CreateNode(e);
foreach link l ∈ Links do
    if (TypeLink(l)=contribution or TypeLink(l)=correlation or
    TypeLink(l)=dependency then
        CreateNode(l);
        if (Source(l) and Destination(l) are within the same actor)
        then
            Create an intra-actor dependency link from Destination(l)
            to Source(l);
            Create an intra-actor dependency link from Destination(l)
            to l;
        else
            Create an inter-actor dependency link from Destination(l)
            to Source(l);
            Create an inter-actor dependency link from Destination(l)
            to l;
        end
    end
    if (TypeLink(l)=decomposition) then
        if (no node is created for l) then
            CreateNode(l);
        end
        if (Source(l) and Destination(l) are within the same actor)
        then
            Create an intra-actor dependency link from Destination(l)
            to Source(l);
        else
            Create an inter-actor dependency link from Destination(l)
            to Source(l);
        end
    end
end
```

Fig. 5. Constructing the GMDG graph

the value of a variable V at a point of interest, called the *slicing criterion*. The application of slicing has been extended to other software artifacts such as requirements and design models, formal specifications, software architectures, etc. In what follows, we extend the application of slicing to GRL models. In our CIA approach, we are interested in obtaining the GRL elements that are impacted by a given modification represented by the *slicing criterion*.

Definition 4 (GRL Slicing Criterion): Given a GRL model, a slicing criterion may be either a GRL intentional element, a GRL link, or a collapsed actor.

The proposed slicing algorithm (see Figure 6) is based on a backward traversal of the GMDG. It starts with the localization of the node corresponding to the slicing criteria. Next, it follows all incoming links and marks all encountered nodes. The procedure stops when all reached nodes have no incoming edges. Encountering inter-actor dependency links is an indication that the proposed change would affect other actors.

The GMDG marked nodes are then mapped back to the original GRL model, representing all GRL constructs impacted by the proposed change.

```
Input  : A GMDG + a slicing criterion (SC)
Output: A set of marked GMDG nodes + whether the impact is spread
        to other actors (ImpactingOtherActors)
Locate SC;
Mark node SC;
currentNode= SC;
ImpactingOtherActors = false;
forall the incoming links to SC do
    if (incoming link is inter-actor) then
        ImpactingOtherActors = true;
    end
    Follow the link;
    Mark the reached node;
    currentNode ⟵ reached node;
    recursively call the algorithm with GMDG and currentNode as
    input
end
```

Fig. 6. GMDG Slicing Algorithm

IV. Case Study: Undergraduate Student Involvement in Research Activities

In this section, we apply our proposed CIA approach to a GRL model describing undergraduate student involvement in research activities (see Figure 7). The model involves two actors (Professor and Undergraduate Student) and describes one explicit dependency stating that "In order to achieve student satisfaction with their participation in university research projects, undergraduate students depend on professors to ensure active involvement of students in their research projects". Research opportunities may take one of the following two forms: (1) programming duties and (2) experiments and data collection. These duties are described as two professor tasks (i.e., "Assign programming duties to undergraduate students" and "Assign experiments and data collection to undergraduate students") contributing positively (i.e., using two GRL *help* contributions) to the softgoal "Active involvement of undergraduate students in research projects". The latter softgoal hurts (i.e., using the *hurt* contribution type) the completion of critical projects having tight deadlines. Student satisfaction with research activities is modeled as a softgoal "Students satisfied with their participation in university research projects" and it is subject to getting an academic credit for their performed tasks (i.e., goal "Receive academic credits for research activities") and getting a financial compensation (i.e., goal "Receive financial compensation for research activities") from professors.

Figure 8 illustrates three changes to be implemented in the original GRL model:

1) *Change1*: Addition of a new task "Assign mechanical and electrical assembly tasks to undergraduate students" contributing positively (i.e., *SomePositive* type) to the softgoal "Active involvement of undergraduate students in research projects".
2) *Change2:* Replace the *And* by an *Or* decomposition.
3) *Change3:* Deletion of task "Assign programming duties to undergraduate students" and its corresponding *help* contribution.

 ISBN: 978-1-61208-449-7

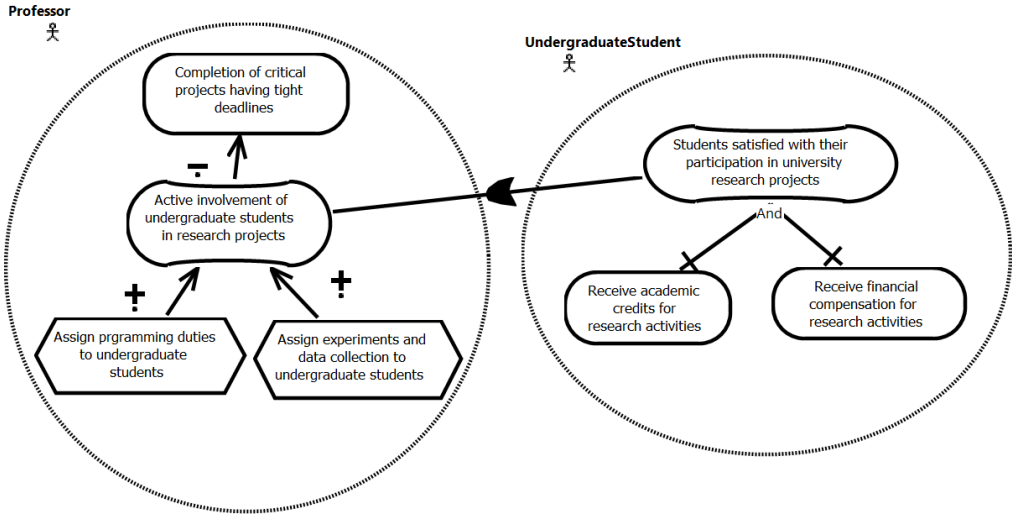
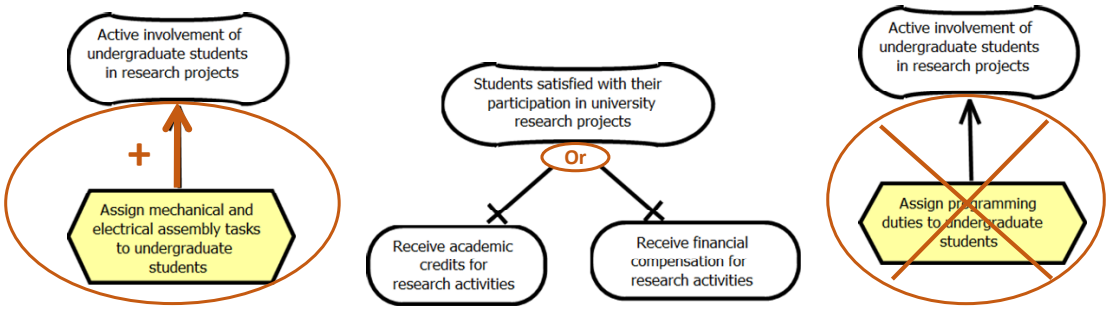


Fig. 7. GRL model describing undergraduate student involvement in research activities



(a) Change1: Add a new task and a new help contribution

(b) Change2: Replace the *And* by an *OR* decomposition

(c) Change3: Delete a task and its corresponding contribution

Fig. 8. Three planned changes to the GRL model

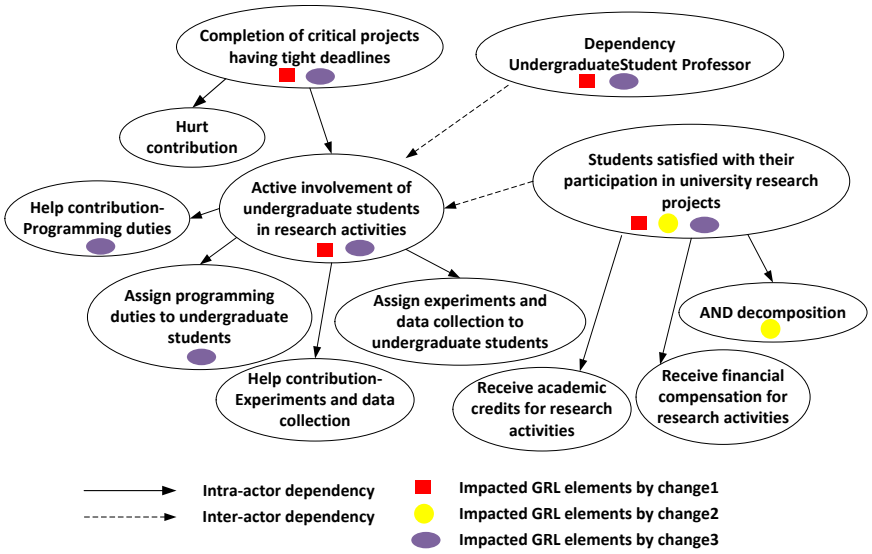


Fig. 9. GRL Model Dependency Graph corresponding to the GRL model in Figure 7

 ISBN: 978-1-61208-449-7

Figure 9 illustrates the marked GMDG corresponding to the GRL model in Figure 7 and the three proposed changes of Figure 8.

V. Comparison with related work

There have been a number of studies on goal models that establish traceability links between the requirements and the system design. Lamsweerde [6] has proposed an approach to derive software architectures from a system goal model using heuristics. The approach assigns responsibilities for achieving goals to their corresponding components and establish connections among them. Yu et al. [7] have proposed a technique for generating a highly versatile software design from goal models. Their technique transforms goals into components and determines component connections from AND/OR-refinement links. Lee et al. [8] have proposed a change impact analysis approach using a goal-driven traceability-based technique. The authors have used traces among goals and use cases to analyze requirements changes. Goals and use case are connected via three different traceability relations (evolution, dependency, and satisfaction), which are stored in a design structure matrix. Impacted entities can then be determined by applying a reachability analysis on the matrix. While these approaches establish traceability with other artifacts, our proposed approach focus on the impact of changes in the goal model itself. Furthermore, our approach can be combined with approaches like the ones presented in [6] [7] [8] by adding traceability links from the GRL elements to other artifacts.

Some studies have proposed ways to support developers in making requirements changes in goal models. Ernst et al. [9] have introduced the notion of a Requirements Engineering Knowledge Base (REKB) for maintaining a requirements model. The authors explore the case where unanticipated changes occur to the requirements of an operational system, such as a new law coming into effect, or adding new features suggested by the marketing team. Our proposed approach is different from this respect as we conduct change impact analysis on goal models once requirements changes have been identified. Cleland-Huang et al. [10] have introduced a probabilistic approach to manage traceability links for non-functional requirements. Non-functional requirements and their dependencies are modeled with a Softgoal Interdependency Graph (SIG). Designers can then analyze the impact of changes by retrieving links between classes affected by changes in a softgoal interdependency graph. While Cleland-Huang et al. use the interdependence of non-functional requirements, our proposed approach is based on the intrinsic structure of the goal model and does not consider the type of requirements. Nakagawa et al. [11] proposed a process of elaboration for goal models, expressed in KAOS [12], that extracts a set of control loops from the requirements descriptions. These control loops are considered to be independent components, hence, preventing the impact of a change from spreading outside them. Our proposed approach allows for the identification of effects spreading between GRL actors.

VI. Conclusions and future work

We have proposed an approach to change impact analysis that allows requirements engineers to assess the possible impact of early changes in goal-oriented models. The proposed approach identifies whether the proposed changes are propagated to external actors. Furthermore, we have extended the use of the well-known technique of program slicing to GRL Model Dependency Graphs, that are derived from GRL models to describe model dependencies. As a future work, we plan to combine our approach with existing GRL goal satisfaction evaluation strategies, in order to have a precise assessment of the magnitude of a given change.

References

[1] R. S. Arnold, Software Change Impact Analysis. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[2] S. Lehnert, "A taxonomy for software change impact analysis," in Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution, ser. IWPSE-EVOL '11. New York, NY, USA: ACM, 2011, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/2024445.2024454

[3] ITU-T, "Recommendation Z.151 (10/12), User Requirements Notation (URN) language definition, Geneva, Switzerland," Genève,Switzerland, 2012. [Online]. Available: http://www.itu.int/rec/T-REC-Z.151/en

[4] M. Weiser, "Program slicing," in Proceedings of the 5th International Conference on Software Engineering, ser. ICSE '81. Piscataway, NJ, USA: IEEE Press, 1981, pp. 439–449. [Online]. Available: http://dl.acm.org/citation.cfm?id=800078.802557

[5] J. Hassine and M. Alshayeb, "Measurement of actor external dependencies in GRL models," in Proceedings of the Seventh International i* Workshop co-located with the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), Thessaloniki, Greece, June 16-17, 2014., ser. CEUR Workshop Proceedings, F. Dalpiaz and J. Horkoff, Eds., vol. 1157. CEUR-WS.org, 2014. [Online]. Available: http://ceur-ws.org/Vol-1157/paper22.pdf

[6] A. van Lamsweerde, "From system goals to software architecture," in Formal Methods for Software Architectures, ser. Lecture Notes in Computer Science, M. Bernardo and P. Inverardi, Eds. Springer Berlin Heidelberg, 2003, vol. 2804, pp. 25–43.

[7] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. S. P. Leite, "From goals to high-variability software design," in Proceedings of the 17th International Conference on Foundations of Intelligent Systems, ser. ISMIS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1786474.1786476

[8] W.-T. Lee, W.-Y. Deng, J. Lee, and S.-J. Lee, "Change impact analysis with a goal-driven traceability-based approach," International Journal of Intelligent Systems, vol. 25, no. 8, 2010, pp. 878–908. [Online]. Available: http://dx.doi.org/10.1002/int.20443

[9] N. Ernst, A. Borgida, and I. Jureta, "Finding incremental solutions for evolving requirements," in 19th IEEE International Requirements Engineering Conference (RE), Aug 2011, pp. 15–24.

[10] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, "Goal-centric traceability for managing non-functional requirements," in Proceedings of the 27th International Conference on Software Engineering, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 362–371. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062525

[11] H. Nakagawa, A. Ohsuga, and S. Honiden, "A goal model elaboration for localizing changes in software evolution," in Requirements Engineering Conference (RE), 2013 21st IEEE International, July 2013, pp. 155–164.

[12] A. van Lamsweerde, "Requirements engineering: from craft to discipline," in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2008), Atlanta, Georgia, USA, 2008, pp. 238–249.

 ISBN: 978-1-61208-449-7