A CHANGE IMPACT ANALYSIS APPROACH TO USER REQUIREMENTS NOTATION (URN) MODELS

BY

HASAN SALIM OMAR AL-KAF

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SOFTWARE ENGINEERING

DECEMBER 2017

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **HASAN SALIM OMAR AL-KAF** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SOFTWARE ENGINEERING**.

Thesis Committee

[illegible]

Dr. Jameleddin Hassine (Adviser)

(Co-adviser)

[illegible]

Dr. Mohammad Alshayeb (Member)

[illegible]

Dr. Sami Zhioua (Member)

(Member)

[illegible]

Dr. Khalid Al-Jasser
Department Chairman

[illegible]

Dr. Salam A. Zummo
Dean of Graduate Studies

عمادة الدراسات العليا
جامعة الملك فهد للبترول والمعادن
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
1963
DEANSHIP OF GRADUATE STUDIES

14/2/18

Date

I dedicate all what I have done to my dear parents, Salem and Maryam, my love Zainab, brothers, and sweet sister.
Words cannot express how much I'm grateful of your presence in my life, and I'm thankful for all what you have given to me.

ACKNOWLEDGMENTS

First, Im grateful to ALLAH almighty for everything in my life. Without his guidance and blessing, this thesis would not have been completed.

I would like to express my gratitude and appreciation to my advisor and mentor Dr. Jameleddine Hassine for his excellent guidance, encouragement, caring, and providing me with all needs for doing my research. A special thanks to my committee members Dr. Mohammad Alshayeb, and Dr. Sami Zhioua for their support and valuable comments.

I would like to thank and appreciate the Hadhramout Establishment for Human Development for their financial support during my academic journey.

Finally, Id like thank my dears parents, brothers, and sister for their support and their prayers. They were always encouraging and motivating me to do the best in my life and achieved all my goals till now. Also, I would like to thank my best friends (Ahmed, Manaf, Omer, Taha). Thank you

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME:	Hasan Salim Omar Al-Kaf
TITLE OF STUDY:	A Change Impact Analysis Approach to User Requirements Notation (URN) Models
MAJOR FIELD:	Software Engineering
DATE OF DEGREE:	December 2017

Requirements tend to change over time in response to the evolving needs of stakeholders, technologies advances, changes in business environments and global competition. Therefore, there is a need for mechanisms to identify and analyze the potential impact of the proposed changes in requirements artifacts before the actual changes are implemented. The User Requirements Notation (URN), an ITU-T standard, is a modeling language that is intended for the elicitation, analysis, and validation of high-level requirements. The URN notation combines two complementary sub-languages: the Goal-oriented Requirement Language (GRL) for modeling actors and their intentions, and the Use Case Maps (UCM) language for describing functional scenarios bound to architectural components. In this paper, we propose a Change Impact Analysis (CIA) approach to User Require-

ments Notation (URN) models. Given a suggested modification within a given GRL or UCM model, our approach allows for the identification of all potentially impacted elements within the selected model, as well as across all UCM and GRL models linked to it through URN Links. The proposed URN-based CIA approach is implemented as a feature within the Eclipse-based jUCMNav framework. We demonstrate the applicability of our approach using a URN mock system and three publicly available real-world URN specifications. Furthermore, we have conducted an empirical study to demonstrate that the proposed URN-based CIA approach improves the ability to identify impacted elements, as part of a requested URN specification change.

CHAPTER 1

INTRODUCTION

During software development life-cycle, requirements changes are inevitable in order to fulfill changing stakeholders goals, accommodate changes in business environments, meet technologies advances, and to respond to competition. Requirements models are often the first artifacts created during early stages of the software development life-cycle. Requirements models are deemed to evolve and grow over time as they go through many necessary modifications in order to meet customers' needs. However, one of the major issues is that seemingly small requirements changes can ripple throughout the system to have major unintended effects elsewhere [1]. Therefore, there is a need for techniques to identify the impact of requirements changes in order to understand and assess how such changes propagate through the requirements, so that informed decisions can be made; hence maintaining consistency and facilitating the successful evolution of software. One of the most efficient techniques is Change Impact Analysis (CIA) [2], which is defined as "the activity of identifying the potential consequences, including side effects and ripple effects, of a change, or estimating what needs to be modified to accomplish a change before it has been made". CIA helps maintainers estimate the extent and cost of the effect of changes and allows them to evaluate and select a suitable solution from a set of potential alternatives. Change impact analysis techniques have been applied to source code [3], requirements [4, 5, 6], architectural models [7, 8], and to different combinations of code, architecture and requirements [9, 10].

The User Requirements Notation (URN) [11], an ITU-T standard, is a visual modeling language that supports the elicitation, analysis, and validation of early requirements. URN describes visually and in one unified language, goals and functional scenarios, and the links between them. It offers two sub-languages: (1) the Use Case Maps (UCM) language for describing high-level scenarios and architectures and (2) the Goal-oriented Requirement Language (GRL) for modeling stakeholders intentions and their business goals. Modeling goals and functional scenarios are complementary and would help uncovering additional goals and scenarios. Hence, such combination will contribute to the precision and completeness of requirements. URN offers a mechanism to link any two URN model elements (called URN links), establishing traceability between GRL and UCM models, which help us achieve completeness and consistence analysis [12].

1.1 Research Motivation

The main motivation of this thesis is to identify the change impact analysis to User Requirements Notation (URN), in order to help maintainers, manage URN models by analyzing the impact of changes on both UCM and GRL models. Developing an efficient change impact analysis algorithms for the URN language would help software engineers understand the impact of a change prior to performing a maintenance task, hence, increasing their productivity and reducing the cost of typical maintenance tasks. In particular, we are interested in understanding and capturing how changes propagate through URN model (i.e. between GRL and UCM models and vice versa).

In this thesis, we develop a new CIA feature to target the URN models and we integrate CIA activity within, the eclipse-based, jUCMNav [13] tool.

1.2 Thesis Objectives

The main objective of our research work is to propose a Change Impact Analysis (CIA) approach to User Requirements Notation (URN) models. Our approach allows the analyst to identify the all potentially impacted elements within GRL and UCM models, as well as across all UCM and GRL models linked to them through URN Links. To achieve this objective, our work is provided the following:

- It provides a unified URN-based approach to change impact analysis that combines GRL and UCM languages. It shows how changes are propagated

(1) within a GRL model, (2) within a UCM model, and (3) how changes are propagated across models (i.e. from GRL to UCM and vice versa) through traceability links (i.e. URN links).

- It provides an implementation of the proposed change impact analysis within the jUCMNav tool [13].
- It demonstrates the applicability of the proposed approach using a URN mock system (covering all language constructs) and three real-world publicly available URN specifications.
- It validates empirically the usefulness of the proposed CIA approach in improving the understandability of URN models and facilitating the identification of the impacted URN elements, as part of a suggested maintenance task.

1.3 Thesis Approach

The research methodology that followed in this research work is a combination of algorithm design, analysis, empirical evaluation, and empirical validation. Our methodology consists of the following steps:

- Design and implementation of the CIA algorithm for *GRL* Models.
- Design and implementation of the CIA algorithm for *UCM* Models.

- Design and implementation of the CIA algorithm from GRL to UCM models and vice versa (i.e. through URN Links).
- Validate the proposed approach using publicly URN specification available and a mock system.
- Conduct an experiment to test the proposed work.
- Evaluate the usefulness of the proposed approach.

1.3.1 Evaluation the proposed CIA approach

We evaluate our verification methodology through its application to many specifications of different types of GRL model and UCM models by conducting an experiment to demonstrate the evidence that supports the benefits of change impact analysis feature in facilitating both the correctness and the comprehension of URN models.

1.3.2 Empirical Validation

In addition to evaluation, we validate our CIA approach by conducting an empirical experiment evolving 10 participants to assess our change impact analysis approach to demonstrate the applicability of the proposed approach using a URN 3 publicly available URN-based case studies of different sizes, complexity, and feature.

1.4 Contributions

This thesis has the following contributions:

1.4.1 Contribution 1: CIA algorithm to GRL Models

Design and implementation of the CIA algorithm to GRL Models.

- The proposed change impact analysis approach allows maintainers and analyst to understand how a change in a GRL model is propagated within the model itself (e.g., between actors of the model) and across other GRL models (i.e., GRL to GRL propagation) through URN Links. Furthermore, the proposed approach allows for the identification of the potentially impacted GRL evaluation strategies as a result of a proposed change.

- It provides a prototype tool that automates the proposed GRL-based change impact analysis approach. The prototype is implemented as a feature within the jUCMNav [13] tool and is publicly available.

  The CIA feature can be downloaded from https://github.com/JUCMNAV/projetseg/tree/grl

- Published paper under the title: An Automated Change Impact Analysis Approach to GRL Models [14].

  Source: https://link.springer.com/chapter/10.1007/978-3-319-68015-6_10

1.4.2 Contribution 2: CIA algorithm to UCM Models

Design and implementation of the CIA algorithm to UCM Models.

- It provides a change impact analysis to UCM Models and how changes are propagated within a UCM Model, and how changes are propagated across models (i.e. UCM to UCM propagation) through URN Links.
- It provides a prototype tool that automates the proposed UCM-based change impact analysis approach. The prototype is implemented as a feature within the jUCMNav [13] tool and is publicly available.
- It provides an algorithm that identifies the URN Links between URN elements within model through URN Links. The link might be between UCM to UCM or UCM to GRL.

1.4.3 Contribution 3: Experimental Evaluation and Validation

Assessment of the proposed work. We evaluated our proposed work using 3 publicly available URN-based case studies of different sizes, complexity and features, and one mock system model that covers all URN constructs. The experiments evaluation showed that change impact analysis approach can be applied to URN-based models of different sizes, complexity, and features. Then, we conducted some empirical experiments to assess our change impact analysis approach. The results show that the approach contributes to the correctness of the URN model and reducing the time consuming to identify the impact of change.

1.5 Thesis Organization

The rest of the thesis is organized as follows. In chapter 2, we provide a background needed to cover this research and literature review. Chapter 3 describes the GRL change impact analysis approach. In chapter 4, we describe the UCM change impact analysis approach. In chapter 5, we provide the details of experimental works and results. Then, in chapter 6, we discuss the benefits and limitations of proposed work. Finally, we conclude and suggest future work in chapter 7.

CHAPTER 2

BACKGROUND

Bohner and Arnold [2] identified the CIA as the process of identifying the potential consequences of a change, or estimate what needs to be modified to accomplish a change. The change request is considered as an input to Change Impact Analysis process. The change request has done by the stakeholders, e.g., product managers, customers, or users.

In the literature below, several source code based CIA techniques have been proposed in order to help the understandability of software, debugging, or regression test. Based on program source code, Li et al. [3] conducted a survey of change impact analysis techniques. Based on this survey, techniques can be divided into 4 main categories:

- Techniques based on traditional static program analysis techniques involving the analysis of dependency graph (e.g., through reachability analysis) [15, 16, 17].
- Techniques based on analysis of the information collected during the pro-

gram execution [18, 19, 20].

- Techniques based on mining information from the software repositories [21, 22].
- Techniques based on measurement of coupling (e.g., structural, conceptual, etc.) [23, 24, 25].

Later, the change impact analysis research has extended to other artifacts such as design, requirements, and testing. A taxonomy for software change impact analysis was developed by Lehnert [26] and a comprehensive literature review [27] of 150 studies was conducted that related to change impact analysis of source code, architecture [28, 29], miscellaneous artifacts (e.g., configuration files, bug trackers, documentation) [30], and requirements models [31, 32, 33, 34]. The traceability links between system design and requirement have been investigated by many researchers.One of them, a new technique has suggested for generating a highly useful software design from foal models by Yu et al. [35]. This technique converts all goals into components and determines their connections between all components from AND OR-refinement links. Also, Lee et al. [33] have based their theory using a goal-driven traceability-based technique in a CIA approach.

The researchers have utilized traces between goals and use cases to analyze all proposed changes of requirements. Using cases and utilization trace among goals are linked via three traceability relations are evolution, satisfaction, and dependency. This traceability has been Stored in a design structure matrix, then the impacted entities can be determined after performed a reachability analysis on structured matrix. Another study by Lamsweerde [36] has established his theory to extract software architectures from a system goal model by heuristics which means that defines tasks for achieving goals to their corresponding components and establish connections among them. Some researchers have been recommended by studies to support developers to give more attention to requirements changes in terms of goal models. Ernst et al. [37] for keeping a requirements model maintainable, they have proposed the notion of a Requirements Engineering Knowledge Base (REKB). The authors discover unanticipated modification that might occur in the operational system requirements. For instance, adding a new feature, or add a new law coming to effects by the team. The difference between Ernst et al. [37] and our proposed approach is that we apply CIA analysis once there is any change on goal models. Based on that known requirement changes. Cleland-Huang et al. [38] have presented new approach-based probabilistic to non-functional needs by managing traceability links. Soft-goal Interdependency Graph (SIG) constructs by shaping Non-functional needs and their dependencies. Developers can then analyze the effect changes by recovering all link have been changed in a SIG graph to affected classes. On the other side, Cleland-Huang et al. ignore the Non-functional needs and their dependencies and replaced with the non-functional interdependence requirements, our proposed CIA relies on the core structure of the goal models and regardless the type of requirements. Nakagawa et al. [39] provided extra details for goal models, expressed in KAOS [40], whereas from the requirement descriptions, a set of control loops have been explored. In recent work [41], the author has investigated the use of slicing technique to analyze the propagation of changes in GRL models by using GRL. The proposed technique extracted the dependency from the GRL model using a GRL Model Dependency Graph (GMDG). GMDG consider two types of dependencies which are intra- and inter- actor dependencies. By proposed change we can identify the constructs that are impacted. They apply slicing to the GMDG model. The initial results were promising. Furthermore, in the early work [31], both slicing and dependency analysis were applied at the Use Case Map (UCM) level in order to grasp the effect of requirements changes. This approach does not consider neither the UCM data flow model nor the URN links to GRL models. Although many studies have been done in the maintenance of URN-based models, none of them has provided such techniques to assess the impact of changes in both GRL and UCM models by using URL links that link any two URM model elements and establish traceability between GRL and UCM.

Change impact analysis approaches can be divided into (3) classifications: (1)dependency impact analysis, (2)traceability impact analysis, and Experimental impact analysis [42, 2]. The impact analysis techniques based on dependency analysis [43, 44, 45, 46, 47, 48, 49] attempt to assess the effects of change on requirements.

2.1 URN in a Nutshell

The User Requirements Notation (URN) [11], an ITU-T standard, is a visual modeling language that supports the elicitation, analysis, and validation of early requirements. URN describes visually and in one unified language, goals and functional scenarios, and the links between them. It offers two sub-languages: (1) the Use Case Maps (UCM) language for describing high-level scenarios and architectures and (2) the Goal-oriented Requirement Language (GRL) for modeling stakeholders intentions and their business goals. Modeling goals and functional scenarios are complementary and would help uncovering additional goals and scenarios. Hence, such combination will contribute to the precision and completeness of requirements. In a requirement engineering process, Liu and Yu [50] found that there is a link between goals a scenario. Goal oriented model helps the analyst to find an important scenario of goal, and the scenarios help the analyst to find out new goals. Weiss and Amyot [51] illustrated the benefits of combining GRL with UCM for modeling.

2.1.1 GRL in a Nutshell

The Goal-oriented Requirement Language (GRL) [11], part of ITU-T's User Requirement Notation (URN) standard, is a visual modeling notation that is used to model intentions, business goals, functional and non-functional requirements (NFR). A GRL goal model is a graph of intentional elements, that optionally reside within an actor. Actors (illustrated as Actor) are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available, and softgoals to be satisfied [11]. Actor definitions are often used to represent stakeholders as well as systems. A GRL actor may contain intentional elements and indicators describing its intentions, capabilities and related measures.

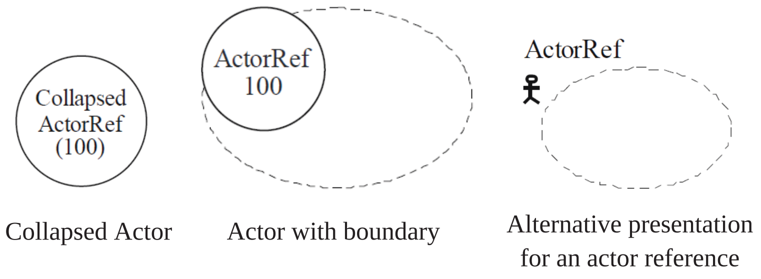
Softgoals (illustrated as) differentiate themselves from goals (illustrated as) in that there is no clear, objective measure of satisfaction for a softgoal whereas a goal is quantifiable, often in a binary way. Tasks (illustrated as) represent solutions to (or operationalizations of) goals or softgoals. In order to be achieved or completed, softgoals, goals, and tasks may require resources (illustrated as) to be available. A GRL indicator (illustrated as) is a GRL element that is used to represent some real-world measurements. An indicator usually convert real-world values in user-defined units into GRL satisfaction values on a standard scale (e.g.[–100, 100]).

Various kinds of links connect the elements in a goal graph. Decomposition links (illustrated as $\longrightarrow\!\!\!\!+$) allow an element to be decomposed into sub-elements (using AND, OR, or XOR). Contribution links (illustrated as $\longrightarrow$) indicate desired impacts of one element on another element. A contribution link has a qualitative contribution type (e.g., Make, Help, SomePositive, Unknown, SomeNegative, Break, Hurt) and/or a quantitative contribution (e.g., an integer value within [–100, 100]). Correlation links (illustrated as $\dashrightarrow$) describe side effects rather than desired impacts. Dependency links (illustrated as $\rightarrow\!\!\blacktriangleright\!\!\!-\!\!-$) model relationships between actors, where intentional elements inside actor definitions can be used as source and/or destination of a dependency link. In this research, we adopt the classification of GRL dependencies introduced in [52] that considers contributions, correlations and decompositions links as implicit dependencies, and dependency links as explicit dependencies.

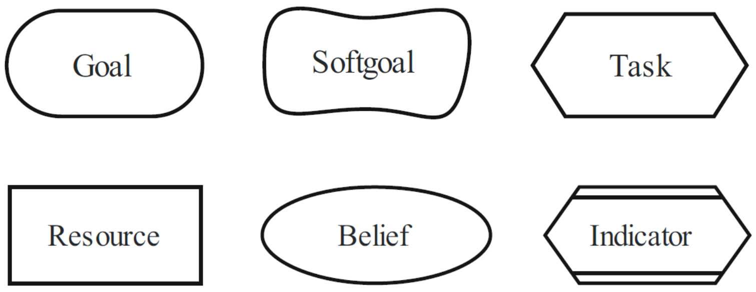
Initial satisfaction levels, which can be quantitative (e.g., within [–100, 100]), or qualitative (e.g., Satisfied, Weakly Satisfied, Denied, Weakly Denied, etc.) of some of the intentional elements constitute a GRL strategy. These initial values (emanating from a contextual or a future situation) propagate to the other intentional elements of the model through the various model links, allowing for the assessment of how high-level goals are achieved and may reveal more appropriate alternative strategies. Finally, URN Links (illustrated as a black triangle symbol ► (source) ◄ (target)) are used to connect a source URN model element with a target URN model element. URN Links model user-defined relationships such as traceability, refinement, implementation, etc. For a detailed description of the GRL language, the reader is invited to consult [11].

2.1.2 UCM in a Nutshell

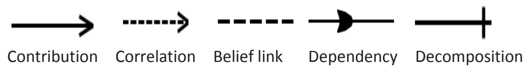
The Use Case Maps (UCM) language, part of the ITU-T User Requirements Notation (URN) standard [11], is a high-level visual scenario-based modeling language. Use Case Maps are used to capture and integrate high-level functional scenarios in terms of causal relationships between responsibilities (×, i.e., the steps within



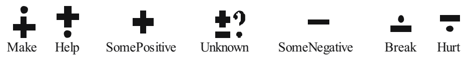
(a) GRL actors



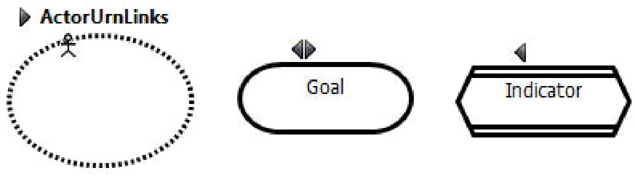
(b) GRL intentional elements



(c) GRL links



(d) GRL qualitative contribution types

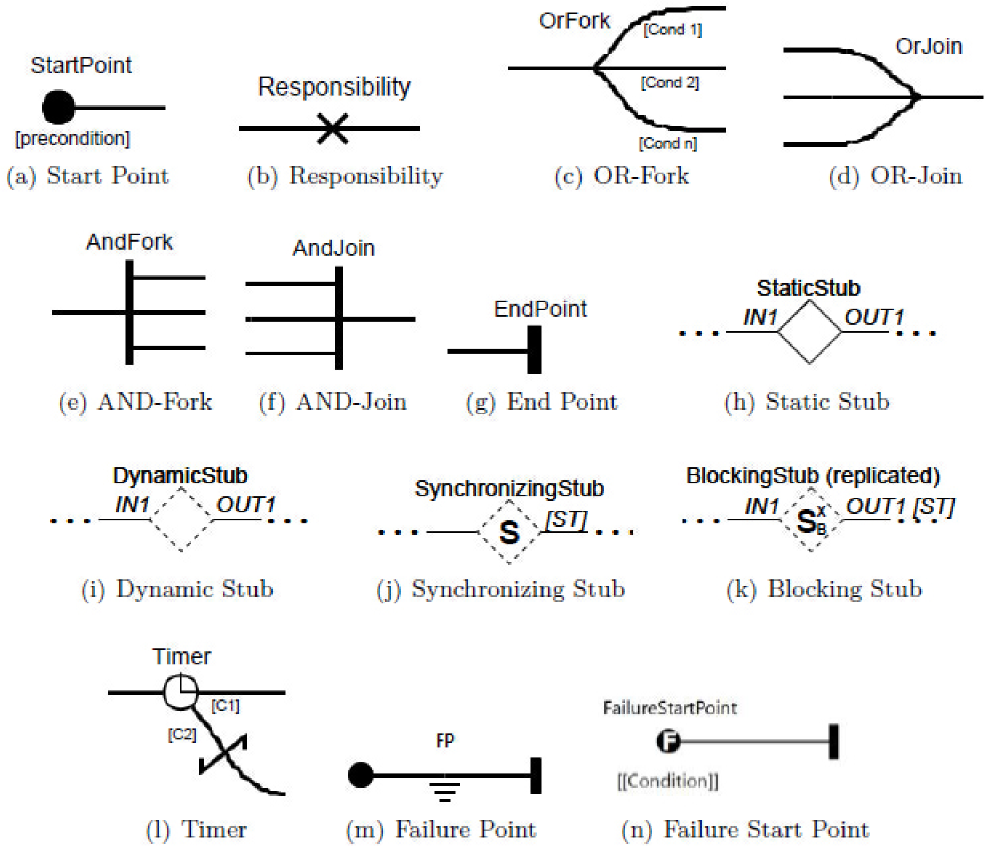


(e) URN links types

Figure 2.1: Goal-Oriented Language Components

a scenario describing operations, tasks, actions, etc.) along paths allocated to a set of architectural components (▭). UCM Scenarios may be integrated sequentially (in a map-like diagram), as alternatives (with OR-forks/joins; ≺/≻), or concurrently (with AND-forks/joins; ⊣/⊢).

For a detailed description of the UCM language, the reader is invited to consult [11]. One of the strengths of UCMs resides in their ability to bind responsibilities to architectural components. Several kinds of UCM components allow system entities (▭) to be differentiated from entities of the environment (♀). Components can be organized hierarchically, i.e., vertical decomposition, through the *component containment* mechanism.



(a) Start Point (b) Responsibility (c) OR-Fork (d) OR-Join

(e) AND-Fork (f) AND-Join (g) End Point (h) Static Stub

(i) Dynamic Stub (j) Synchronizing Stub (k) Blocking Stub

(l) Timer (m) Failure Point (n) Failure Start Point

Figure 2.2: Use Case Maps Basic Constructs

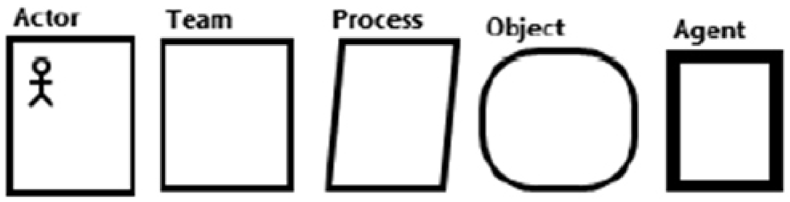


Figure 2.3: Use Case Maps Components

When maps become too complex to be represented as one single UCM, a mechanism for structuring sub-maps becomes necessary. Path details can be hidden in sub-diagrams called plug-in maps, contained in stubs (presented as *diamonds*) on a path. A plug-in map is bound (i.e., connected) to its parent map by binding the in-paths and out-paths of the stub to start points (●) and end points (❙) of the plug-in map, respectively. UCM has a mechanism which allows to define and structure sub-maps. Path details can be hidden in sub-maps called plug-ins contained in stubs (diamonds) on a path. UCM supports four types of stubs: (1) static stub (◇) has at the most one plug-in map that cannot be replicated and that is always selected, (2) dynamic stub (◇) may have many plug-in maps that can be replicated and that are selected according to some selection policy. When the UCM path reaches the dynamic stub, the selected plugin maps of the stub are traversed in parallel, (3) synchronizing stub (◈) is a dynamic stub that in addition synchronizes its plug-in maps before the traversal of the UCM path is allowed to continue past the stub. A synchronization threshold can be defined for each out-path of a stub, (4) blocking stub (◈) is a synchronizing stub that does not allow its plug-in maps to be visited more than once at the same time.

Figure 2.2illustrates the main UCM constructs.

2.2 Program slicing

Program Slicing, introduced by Weiser [53], is a reduction technique used to decrease the size of a program source code by keeping only the statements within a program that are related to the execution of a specific slicing criterion (program location l and the set of variables V, written as (l, V)) specified by the user. The resulting program, called ”static slice”, preserves the semantics of the original program for all possible inputs.

Given a program P and a slicing criterion (l,V), two types of static slices can be produced, backward and forward slices. A backward slice of P with respect to (l,V) consists of all statements and predicates in the program that may affect the value of variables in V at l. A forward slice of P with respect to (l, V) consists of all statements and predicates in the program that may be affected by the value of variables in V at l. Consider the program in Fig. 2.2, that computes the *sum* and the *product* of a set of integer numbers less that a given number *n*. Figure 2.2 illustrates the produced backward slice with respect to the slicing criterion (10, product), while Fig. 2.2 shows the resulting forward slice with respect to the slicing criterion (3, sum).

(a) Original Program -	(b) Backward Slice C=(10, product)	(c) Forward Slice C=(3, sum)
1. $read(n)$	1. $read(n)$	1.
2. $i := 1$	2. $i := 1$	2.
3. $sum := 0$	3.	3. $sum := 0$
4. $product := 1$	4. $product := 1$	4.
5. $while\ (i <= n)\ do$	5. $while\ (i <= n)\ do$	5.
6. $sum := sum + i$	6.	6. $sum := sum + i$
7. $product := product * i$	7. $product := product * i$	7.
8. $i := i + 1$	8. $i := i + 1$	8.
9. $write(sum)$	9.	9. $write(sum)$
10. $write(product)$	10. $write(product)$	10.

Figure 2.4: An example of a program and its corresponding backward and forward static slices

2.3 Forward vs. Backward slicing

Forward and backward traversal can be done at any part of the program, based on a given slicing criterion that indicates to start of the traversal point. The forward slicing technique is highlighting the statements, which is contained a source code, of the original program that may affect by the selected slicing criterion, but backward slicing technique is highlighting all statements that may impact to the selected slicing criterion. Forward slicing techniques help and assist maintainers to predict the portions statements of the program that may affect after performing the maintenance task [54] whereas the backward slices techniques help analyst to identify the portions statements of the program that may contain a bugs. As shown in fig. 2.2, the result of the forward slice with respect to slicing criterion (3, sum). Statement 6 is contributed to the slice due to the right variable of the statement is belong to slicing criterion and statement 9 also is contributed.

CHAPTER 3

GRL CHANGE IMPACT ANALYSIS APPROACH

The content of this chapter is mainly extracted from the paper [14]. Figure 3.1 describes the proposed GRL-based change impact analysis approach. To identify the impact of a change in a GRL model under maintenance, an analyst may select a GRL construct (i.e., an intentional element, an indicator, or a link) to be changed, then specify the type of change (e.g., addition, modification, deletion). Next, the GRL Model Dependency Graph (GMDG) is constructed (see Sect. 3.2), then sliced according to the specified slicing criterion (see Sect. 3.3). GMDG impacted nodes are then identified, mapped back to the original GRL model, and marked with a different color. Finally, impacted evaluation strategies and impacted URN Links are displayed as a GRL Comment construct (see Sect. 3.6).

In what follows, we provide some necessary definitions (adopted and modified from [41]) that are used in the subsequent sections.

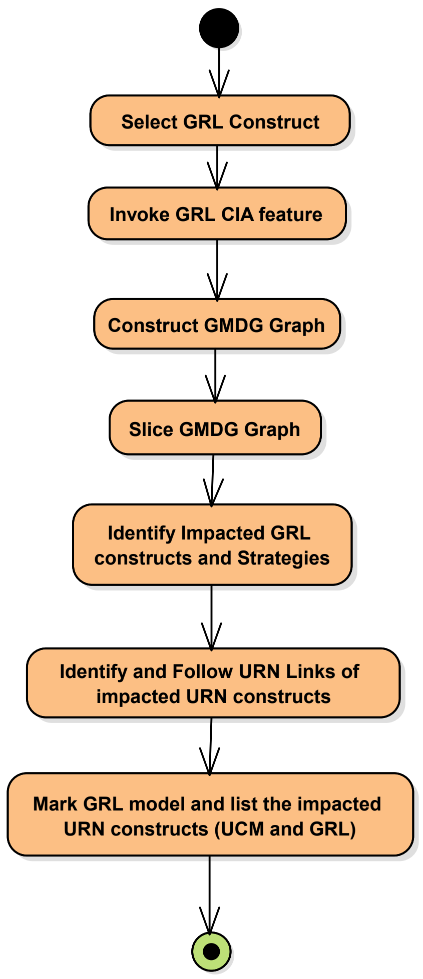

Select GRL Construct
Invoke GRL CIA feature
Construct GMDG Graph
Slice GMDG Graph
Identify Impacted GRL constructs and Strategies
Identify and Follow URN Links of impacted URN constructs
Mark GRL model and list the impacted URN constructs (UCM and GRL)

Figure 3.1: GRL CIA Approach

Definition 3.1 (GRL Model) *We assume that a GRL model* GRLM *is denoted by a 3-tuple: (Actors, Elements, Links), where:*

- Actors *is the set of actor references in the GRL model.*
- Elements *is the set of intentional elements (i.e., tasks, goals, softgoals, resources) and indicators in the GRL model.*
- Links *is the set of links in the GRL model.*

It is worth noting that we don't consider collapsed actors (although they are described in the URN standard [11]), since they are not supported in jUCMNav [13].

Definition 3.2 (GRL Link) *We define a GRL link as (type, src, dest): LinkTypes Elements Elements, where* LinkTypes = {*contribution, correlation, dependency, decomposition*}*), src and dest are the source and destination of the link, respectively.*

Definition 3.3 (GRL Link Access Functions) *Let l=(type, src, dest) be a GRL link. We define the following access functions over GRL links:*

- TypeLink*:* Links $\rightarrow$ LinkTypes*, returns the link type (i.e., TypeLink(l) = type).*
- Source*:* Links $\rightarrow$ Elements*, returns the intentional element source of the link (i.e., Source(l) = src).*

- Destination*:* Links $\rightarrow$ Elements*, returns the intentional element destination of the link (i.e., Destination(l) = dest).*

3.1 Dependencies in GRL Model

The goal-Oriented requirements model illustrates the actors within a large complex system and its requirements, the relationship between systems elements, and goals of organizational. As mention in section 2.1.1, the dependency links provides how a source actors depend on a target actors for an elements/indicator. This relationship shows the reason about how actors depends on other to achieve their goals. we can classify the dependencies in GRL model to implicit or explicit [52]. To model implicit dependencies, we use contribution link (→) Correlation link (⇢), and Decomposition link (➤) . Explicit dependencies are modeled as dependency link. According to [11] required level of details, the explicit dependencies links can be used for many types of configurations. The GRL actor by its definitions, it can be used as destination/ source if explicit dependencies links. In addition, the intentional elements can be used as destination/ source, but with implicit dependencies (Contribution, Correlation, and Dependencies), the actors cannot be used as destination and/or source. Note that the actors overlaps are not allowed in GRL syntax (i.e. share common GRL elements). When a source intentional elements/indicators and a target intentional elements/indicators within the same actors, this called Intra-actor dependencies. When a source intentional elements/indicators and a target intentional elements/indicators bound to different actors, this called Inter- dependencies.

3.2 GRL Model Dependency Graph (GMDG)

In this section, we define the GMDG graph and present the algorithm (Alg. 1) to construct it.

Definition 3.4 (GRL Model Dependency Graph (GMDG)) *A GRL Model Dependency Graph (GMDG) is defined as a directed graph GMDG=(N, E), where:*

- N *is a set of nodes. Each GRL intentional element, indicator, or a link is mapped to a node* n ∈ N*.*
- E *is a set of directed edges. An edge* e ∈ E *represents a dependency between 2 nodes in GMDG and it is illustrated as a solid arrow (*$\longrightarrow$*).*

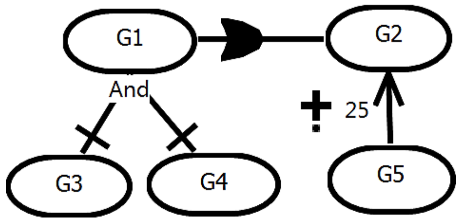
First, for each intentional element, indicator, or a link a new GMDG node is created. Next, depending on the type of the GRL links, GMDG dependency links are created between GMDG nodes (i.e., *CreateDependencyLinkGMDG (e1, e2)* creates a GMDG dependency link from e1 to e2).

Figure 3.2 illustrates a generic GRL model along with its corresponding GMDG graph. Each goal/contribution/decomposition/dependency is represented as a GMDG node. The satisfaction of *G2* depends on the satisfaction of *G5* and the contribution type (*help* in this case), hence, two GMDG links are created: (1)

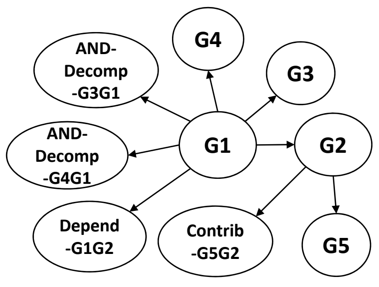
Algorithm 1: Constructing a GRL Model Dependency Graph (GMDG)

```
Procedure Name: ConstructGMDG
Input  : A GRL Model: (Actors, Elements, Links)
Output: A GRL Model Dependency Graph (GMDG)
foreach $e \in Elements$ do
    n= createGMDGNode(e);
end
foreach $e \in Links$ do
    n= createGMDGNode(e);
    if (TypeLink(e) == contribution or TypeLink(e) == correlation or
      TypeLink(e) == decomposition) then
        CreateDependencyLinkGMDG(Destination(e), Source(e)) ;
        CreateDependencyLinkGMDG(Destination(e), n);
    else
        {TypeLink(e) == Dependency}
        CreateDependencyLinkGMDG(Source(e), Destination(e)) ;
        CreateDependencyLinkGMDG(Source(e), n);
    end
end
```

between *G2* and *G5* and (2) between *G2* and *Contrib-G5G2*. Since *G1* is decomposed into *G3* and *G4* (using AND-decomposition), four GMDG dependency links are created: (1) one between *G1* and *G3*, (2) one between *G1* and *G4*, (3) one between *G1* and *AND-Decomp-G3G1*, and (4) one between *G1* and *AND-Decomp-G4G1*. Finally, *G1* depends on *G2*, which is mapped as two GMDG links: (1) one between *G1* and *G2*, and (2) one between *G1* and *depend-G1G2*.



(a) Generic GRL Model



(b) Generic GMDG Graph

Figure 3.2: A Generic GRL model and its corresponding GMDG

3.3 Slicing the GRL Model Dependency Graph

Program Slicing, introduced by Weiser [53] in the early 1980's, is a reduction technique used to decrease the size of a program source code by keeping only the lines within a program that are related to the execution of a specific slicing criterion specified by the user. In order to perform a change impact analysis on GRL models, we extend the concept of program slicing to GMDG graphs [55]. In what follows, we introduce the notion of GRL slicing criterion, then we present the GMDG slicing algorithm (see Alg. 2).

Definition 3.5 (GRL Slicing Criterion) *Let* GRLM *be a GRL model. A slicing criterion SC for GRLM may be either a GRL intentional element/Indicators or a GRL link.*

The slicing of the GMDG (see Algorithm 2) is based on a backward traversal of the GMDG. It requires as input the GMDG graph and the GMDG node that corresponds to the slicing criterion SC. The algorithm starts by adding the GMDG node (called *ImpactedGMDGNode*) to the set of impacted nodes (i.e., *SetGMDGImpactedNodes*). Next, it follows each incoming link leading to *ImpactedGMDGNode* and add its source to *SetGMDGImpactedNodes*. Finally, a recursive call is made by passing the GMDG and the new reached GMDG node.

The resulting set of impacted GMDG nodes (i.e., *SetGMDGImpactedNodes*) is then mapped back to *SetGRLImpactedElements*, the set of the original GRL model elements. The elements within *SetGRLImpactedElements*, along with the

Algorithm 2: GMDG Backward Slicing Algorithm

```
Function Name: SlicingGMDG
Input  : A GMDG + GMDG node corresponding to SC
         (LocationInGMDG(SC))
Output: SetGMDGImpactedNodes
ImpactedGMDGNode = LocationInGMDG(SC);
if ImpactedGMDGNode $\notin$ SetGMDGImpactedNodes then
    AddToImpactedNodes(ImpactedGMDGNode, SetGMDGImpactedNodes);
    if hasIncomingLinks(ImpactedGMDGNode) then
        foreach incomingLink do
            AddToImpactedNodes(Source(incomingLink),
              SetGMDGImpactedNodes);
            GMDGslicingAlg(GMDG, ImpactedGMDGNode);
        end
    end
end
```

impacted elements emanating from following the URN Links (see Sect. 3.4), are then marked in purple color (see examples in Sect. **??**).

3.4 Impact Through URN Links

This step aims at identifying other potential GRL impacted elements by following existing URN Links. A URN Link is used to create a connection between any two URN elements, e.g., intentional element reference/definition, actor reference/definition, link, etc. A URN Link may be defined as follows:

Definition 3.6 (URN Links) *A URN Link is defined as urnl = (type, from, to), where (1)* type *denotes a user-defined URN Link type, (2)* from *denotes the ID of source URN element, and (2)* to *denotes the ID of the target URN element.*

According to [56], the authors defined a rule as a constraints on URN meta model. Algorithm 3 iterates through the set of impacted elements (i.e., *SetGR-*

LImpactedElements) and checks whether these elements are involved in any URN Link, as source (i.e., *from* field) or as a target (i.e., *to* field). Since an impacted element can serve as a source or a target in a URN Link and since one source element can be linked to many target elements and vice versa, we have used two search functions to retrieve the set of elements IDs depending whether we are looking for source or target IDs. (i.e., searchSourceURNLinks and searchTargetURNLinks). The new identified elements are then add to the set *SetGRLImpactedElements*.

Algorithm 3: Excerpt of the algorithm to identify impacted elements emanating from URN Links

```
Function Name: IdentificationOfOverallImpactedElements
Input  : GRL Model + SetGRLImpactedElements
Output: SetGRLImpactedElements
URNLinksList = getAllURNLinks();
foreach $e \in$ SetGRLImpactedElements do
    {Search for target elements IDs when e is defined as source};
    ToElementList = searchTargetURNLinks(e,from,URNLinksList);
    AddToGRLImpactedElements(ToElement, SetGRLImpactedElements);
    {Search for source elements IDs when e is defined as target}
     FromElementList = searchSourceURNLinks(e, URNLinksList);
    AddToGRLImpactedElements(FromElement, SetGRLImpactedElements);
end
```

It is worth to noting that urn links have various links between all URN model elements to establish the traceability between GRL and UCM models, which help us to achieve a completeness and consistence analysis. In case, Intentional Element linked to Actor Defnition or Component Defnition, our proposed CIA algorithm will indicate to all references with their IDs that created by its defnition. Table 3.1 listed all supported elements by our change impact analysis approach. *IER* is a abbreviation of *Intentional Element Reference* and *Link* stands for *Contribution*, *Decomposition*, or *Dependency*.

Intentional Element Reference (**IER**)		
Link Stands for Contribution, Decomposition, or Dependency		
Source Element	*To / From*	**Target Element**
IER	<- >	Intentional Element
IER	<- >	IER
IER	<- >	Link
IER	<- >	Responsibility
IER	<- >	Responsibility Reference
IER	<- >	Actor
IER	<- >	Actor Reference
IER	<- >	Component
IER	<- >	Component Reference
Link	<- >	Intentional Element
Link	<- >	IER
Link	<- >	Link
Link	<- >	Responsibility
Link	<- >	Responsibility Reference
Link	<- >	Actor
Link	<- >	Actor Reference
Link	<- >	Component
Link	<- >	Component Reference

Table 3.1: GRL supported URN links by CIA

3.5 Identification of the Impacted GRL Strategies

Once the set of impacted GRL model elements (i.e., *SetGRLImpactedElements*) is identified, we have to spot all impacted evaluation strategies. Algorithm 4 accepts as input a GRL model and the set of impacted GRL elements (*SetGRLImpactedElements* resulting from applying the GMDG slicing algorithm), and produces the set of impacted GRL strategies (i.e., *SetImpactedStrategies*).

Algorithm 4: Identification of the impacted GRL evaluation strategies

```
Function Name: IdentificationOfImpactedStrategies
Input  : GRL Model + SetGRLImpactedElements
Output: SetImpactedStrategies
SetImpactedStrategies = $\emptyset$;
StrategiesList = getAllStrategies();
foreach strategy $\in$ StrategiesList do
    foreach impactedElement $\in$ SetGRLImpactedElements do
        if PartOfStrategy(impactedElement, strategy) then
            AddToImpactedStrategies(strategy, SetImpactedStrategies) ;
        end
    end
end
```

3.6 jUCMNav GRL-based Change Impact Analysis Feature

Our proposed change impact analysis approach is implemented as a feature [1]1]The CIA feature is publicly available and can be downloaded from `https://github.com/JUCMNAV/projetseg/tree/grl`. within the jUCMNav framework [13], a full graphical editor and analysis tool for GRL models developed as an Eclipse-based plug-in.

To exercise this feature, the user starts by selecting a GRL intentional element, an indicator or a link, then right-clicks to choose from three sub-menu commands: *Addition*, *Deletion*, or *Modification* (see Fig. 3.3). For the addition option, it is required that the analyst adds the GRL construct first then call the feature. The deletion is provided as a separate option because there will be impacted elements due to the loss of connectivity caused by the deletion. It is worth noting that this CIA menu is activated for the supported GRL constructs only.

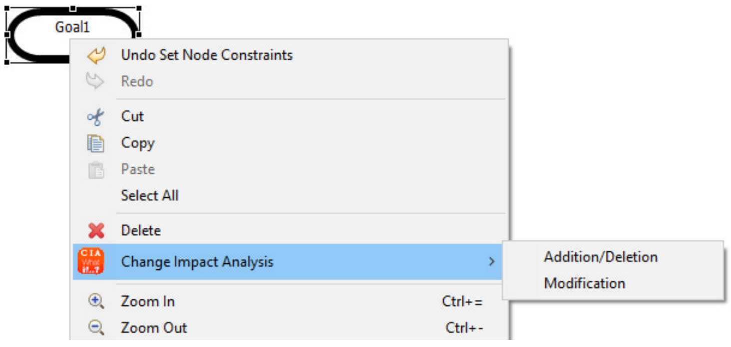


Figure 3.3: GRL CIA included in command menu of jUCMNav framework

If any of the impacted element (marked in purple color (see Fig. 5.5)), is part of a GRL evaluation strategy, the details of the impacted element will appear as a GRL Comment (in gray color) with its name, ID, and the name of strategies it belongs to (see Fig. 5.8(a)). Similarly, information about impacted URN Links, such as SourceID, TargetID, and Type, are also shown in the same GRL Comment box (see Fig. 5.6).

CHAPTER 4

UCM CHANGE IMPACT ANALYSIS APPROACH

In this chapter, we introduce the proposed UCM-based CIA approach. In what follows, we provide some necessary definitions that are used in the subsequent sections.

Definition 4.1 (UCM Slicing Criterion) *A UCM slicing criterion is defined as* SC = *(*targetConstruct*,* SCVariables*), where:*

- targetConstruct *is a simple UCM construct (e.g., responsibility reference (respRef), Or-Fork branch, start point, etc.). A stub cannot be a slicing criterion.*
- SCVariables *is a subset (possibly empty) of variables defined or used within the* targetConstruct*.*

Definition 4.2 (Marked UCM Specification) *Given a UCM Specification* S *and a slicing criterion (SC) for* S. MarkedS *is produced by showing the impacted UCM constructs in* S *(using different colors), with respect to the slicing criterion SC.*

Figure 4.1 describes the main steps of the proposed UCM CIA approach as an activity diagram. An analyst starts by selecting a UCM construct subject to change as part of a maintenance task, then invokes the CIA feature. In case of addition of a new UCM construct, the analyst should add the construct first, then invoke the CIA feature. If the selected construct encloses code, the analyst may select the variables of interest, as part of the slicing criterion (through the GUI shown in Fig. 4.9).

In order to identify the impact of a given change, two main algorithms are executed in parallel (enclosed between *Fork* and *Join* nodes), namely, *UCM Forward Traversal* and *UCM Dependencies Computation.* The forward traversal (see Alg. 6) starts from the selected criterion and visits subsequent constructs on the path, while computing control and data dependencies with respect to the set of selected variables (part of the slicing criterion). The analysis of dependencies leads to the identification of relevant/irrelevant constructs. Next, we follow all URN links associated with the impacted constructs (i.e., relevant constructs) and identify all related UCM/GRL elements. Finally, the UCM is marked (using different colors for the impacted constructs) and a list of all potentially impacted URN elements (GRL and UCM) are displayed as a UCM comment.

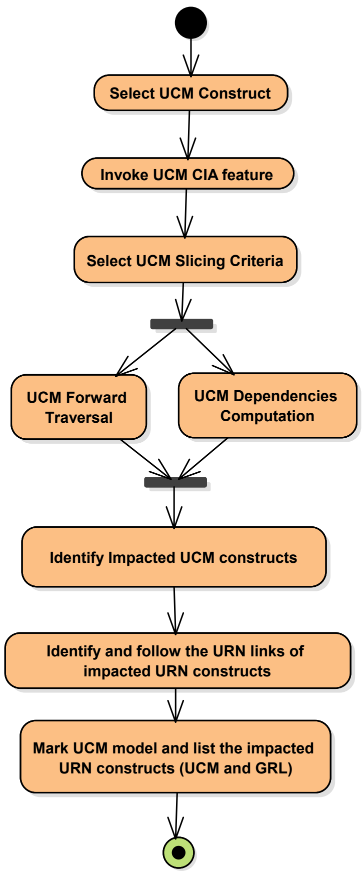

Select UCM Construct
Invoke UCM CIA feature
Select UCM Slicing Criteria
UCM Forward Traversal
UCM Dependencies Computation
Identify Impacted UCM constructs
Identify and follow the URN links of impacted URN constructs
Mark UCM model and list the impacted URN constructs (UCM and GRL)

Figure 4.1: UCM CIA Approach

4.1 UCM construct variables'extraction

Variables are part of the global data model in a URN specification. UCM variables are of several types and can be used within responsibilities (as executable source code), in start points (as a precondition), in end points (as postconditions), within Or-Fork branches (as boolean expressions), in plug-in selection policies for dynamic stubs, and in UCM scenario definitions. URN offers a textual data language with concrete textual syntax allowing for the use of operators from conventional programming languages such as C and Java.

Code expressions vary from assignments, if-else conditions, or blocks of statements (see Table 4.1 for some examples that will be used to explain the different algorithms of our proposed approach). Once the CIA feature is invoked, all variables within the code expression of the selected construct (i.e., the slicing criterion) are extracted and displayed.

4.2 UCM Dependencies computation

In what follows, we define control and data flow dependencies in the UCM context.

Definition 4.3 (UCM Control Flow Dependency) *There is a control flow dependency between two UCM constructs* C1 *and* C2 *if there exists a UCM path that starts at* C1 *and reaches* C2*.* C2 *is said to depend on* C1 *(from a control flow perspective).*

Responsibility name	**Responsibility expression**
resp_1	y:= 0;
resp_2	y := y + 1;
resp_3	t := 0;
resp_4	z := y;
resp_5	z := y +1;
resp_6	t := t + 1;

resp_7	if(i <x)
	{
	i := i + 1;
	n := y;
	}

resp_8	if(y >= x)
	t := t + 1;
	else
	n := y;

resp_9	-

Table 4.1: Responsibilities with their expressions (i.e. source code)

It is worth noting that UCM control flow dependency results from the inherent nature of the UCM notation (i.e., causal paths connecting UCM constructs).

Definition 4.4 (UCM Data Flow Dependency) *There is a data flow dependency between two UCM constructs* C1 *and* C2 *if variables defined (e.g., through assignments) within* C1 *are used within* C2*.* C2 *is said to depend on* C1 *(from a data perspective).*

A responsibility may enclose several code statements. It is sufficient to have a data dependency in one single statement to declare that there is a data flow dependency.

Definition 4.5 (Relevant UCM Construct) *A UCM construct* C *is considered to be relevant, with respect to a slicing criterion* SC = *(*targetConstruct*,* SC-

Variables*), if and only if, there are both a control flow and data flow dependencies between* targetConstruct *and* C*.*

Considering the definition of data flow dependency, it is important to mention that a UCM construct is either relevant (has a control flow dependency and at least has a data flow dependency with respect to one single statement) or irrelevant (i.e., has no control or data flow dependencies).

In our proposed approach, dependencies are calculated on the fly while performing forward traversal of the UCM model.

Algorithm 5 describes the steps of computing and analyzing the extracted code from UCM constructs. The algorithm is not calculated the dependencies when (1) the selected slicing criteria either has not enclosed expression (empty-coded) or (2) it has an expression and the user did not select variables as part of the slicing criterion (i.e., empty SCVariables). In this case, the entire path located after the slicing construct will be considered as impacted.

The expression must be read top-down to extract the right side (i.e. assignment) and left side (i.e. declaration). The algorithm requires as input the code expression and SCVariables which a set of variables within the slicing criterion in order to compute the dependencies. It starts by initializing the relevantVariablesSet which is a set of relevant variables. Then, we check if Construct(SC) does not have a SC Variables, the construct will be stated as a relevant and the entire located path after Construct(SC) will be considered as impacted. Otherwise, if the expression is not empty, we create a function CreateStack that takes

Algorithm 5: UCM Construct Relevancy

```
Procedure Name: DetermineConstructRelevancy
Input  : expression:String {not empty source code}
         SCVariables: Set of variables {variables within the slicing criterion}
Output: relevant: Booelan { true if relevant otherwise irrelevant UCM
         construct}
relevantVariablesSet: Set of variables = ∅;
{set of relevant variables}
CleanExpression(expression);
{Remove comments from expression}
relevant = false;
if (isEmpty(SCVariablesListOf(Construct(SC)))) then
    relevant = true;
    {entire path located after the Construct(SC) will be considered as
     impacted}
else
    if (not(isEmpty(expression))) then
        expressionStatements: Stack (string)= CreateStack (expression);
        {Decompose the expression and store its statements as a stack}
        if (not(isEmpty(SCVariables))) then
            relevantVariablesSet = SCVariables;
            if (currentNode == Construct(SC)) then
                add (relevantVariablesSet, definedVariable (expression,
                 SCVariables));
                {add the newly defined variables to the set of relevant variables}
                pop(expressionStatements);
                relevant = true;
            end
            while (not(isEmpty(expressionStatements))) do
                if (relevantVariablesSet is included in the right side of
                 top(expressionStatements)) then
                    add (relevantVariablesSet, definedVariable
                     (top(expressionStatements), relevantVariablesSet));
                    relevant = true;
                else
                    relevant = false;
                end
                pop(expressionStatements);
            end
            add (SCVariables, relevantVariablesSet);
        else
            {Construct does have an expression, but SCVariables is empty}
            relevant = false;
        end
    end
end
```

as input an expression and generates a stack of statements (i.e. expressionStatements) ignoring all if/while/for statements (using variables). For once, if the selected node equal Construct(SC), the definedVariable will be added to relevantVariablesSet with respect to SCVariables. Next step is by checking whether the expressionStatements is empty or not, if not, the definedVariable will be added to relevantVariablesSet. The dependencies will be computed with relevant variables by checking the variable(s) resides at the right side of the statement. If the variables belong to SCVariables, the left variable of the statement will be added to relevantVariablesSet and the statement is relevant. Then, the expressionStatement will be popped from the stack. In case there is no right variable in a statement and the left variable side belongs to criterion variable list, then the left variable will be removed from the list and it is not relevant because the default value is changed.

In figure 4.4(b) For example, we applying CIA feature with respect to SC= (resp_1, y). First, we add a set of relevant variables to a relevantVariablesSet list of the responsibility (i.e. resp_1). Then, we generate a stack of statements with respect to current responsibility. Next, by checking the expressionStatments if it is not empty, if not, we add the definedVaraible y of the statement (i.e. y:=0;) to the relevantVariablesSet, at the same time we pop it from the stack, and add the responsibility to the global relevantNodes list. Same procedure will be applied with responsibility (i.e. resp_4). In the last responsibility (i.e. resp_8), the expressionStatements stack contained two extracted statements (i.e. t:=t+1; and n:=y;). We check the top expression (i.e. t:=t+1;) with regard the SCVariables (i.e. y), the y is not located in the right side of the top expression, so it will not be added to SCVariables and it will be popped from the expressioStatments stack. But the last statement in the stack, it will be revenant and it will be added to the global relevantNodes list, because the expressionStatement (i.e. n:=y;) is relevant with respect to the SCVariables (i.e. y) which is resided in the right side.

4.3 Forward Traversal Algorithm

Algorithm 6 illustrates the main steps of the UCM forward traversal algorithm. It accepts as in input a startLink which is a successor link of the UCM construct, if the UCM construct is a responsibility, startPoint, or-Join, and-Join, or Timer, startLink will be the direct successor link of the UCM construct. In addition to startLink parameter, SCVariables are the chosen slicing criterion variables (possibly empty). In case of Or-Fork branches and and-Fork branches, the start link will be the successor link of the branches target. The forward traversal algorithm starts by declaring visitedJoins list and stubs list. The visited Joins are the list of all traversed or-Joins that are used to detect the loop and avoid the infinite loop during forward traversal. Each path has its own visitedJoins list. The stubs list, it is used to store all plug-in maps in order to traverse all of them during forward traversal if exist. In our proposed work forward traversal, each UCM construct is handled separately, and it has own procedure. For that reason, we divided the forward procedure into switch cases for the constructs. The procedure will be terminated once reached to an end point and an empty stub list (i.e. there are no nested plug-in maps).

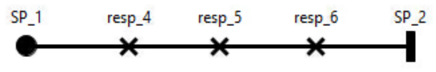
4.3.1 Responsibility References's Procedure

The procedure accepts as input a responsibility reference (respRef), it is handling the respRef during forward traversal by extracting the code, analyzing, and computing the data dependencies. As a result of this procedure, the relevant/irrelevant responsibilities are specified with respect to SCVariables list. in case the respRef does not have code, it will be ignored. Algorithm 7 describes the steps of handling respRef. In order to determine whether respRef is relevant or irrelevant. At the beginning, if the currentRespRef is not a Construct(SC) (i.e. selected UCM construct and its SCVariables), first, we check if the visitedNodes list contained the currentRespRef in order to avoid the loop. Then, we recall DetermineConstructRelevancy function in order to determine the relevancy of currentRespRef, if it is relevant, it will be added to the relevantRespRef list which is used later to color the relevant responsibilities and added to relevantNodes list, otherwise it will be added to the irrelevantRespRef list. Finally, if the currentRespRef is a selected, either it means a loop or it is a SC. First, check whether SCVariables is empty or not. If it is not, we add the currentRespRef to relevantRespRef and relevantNodes, and perform normal determination of relevancy. Otherwise, the currentRespRef will be added to relevantNodes and the rest of UCM constructs come after SC will be added to relevantNodes.

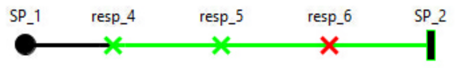
Algorithm 6: Forward Traversal algorithm for CIA approach

```
Procedure Name: ForwardTraversal
Input  : startLink:NodeConnection,
         SCVariables: Set of variables
Output: update SCVariables list and relevantNodes list
currentLink:NodeConnection = startLink;
currentNode:PathNode = getTargetNode(currentLink);
visitedNodes:List(PathNode) = $\emptyset$;
endPoints:List(EndPoint) = $\emptyset$;
visitedJoins:List(PathNode) = getVisitedJoins list;
stubs:List(Stub) = getStubs list;
relevantNodes:List(PathNode) = $\emptyset$;
while (currentNode $\neq$ EndPoint or stubs not empty) do
    switch currentNode do
        case RespRef do
            HandlingRespRef(currentNode) {Alg. 7};
        end
        case Or-Fork do
            HandlingOr-Fork(currentNode) {Alg. 8};
        end
        case Or-Join do
            HandlingOr-Join(currentNode) {Alg. 9};
        end
        case Stub do
            HandlingStub(currentNode) {Alg. 10} ;
        end
        case And-Fork do
            HandlingAnd-Fork(currentNode) (Alg. 11) ;
        end
        case And-Join do
            HandlingAnd-Join(currentNode) (Alg. 12) ;
        end
        case EndPoint do
            HandlingEndPoint(currentNode) {Alg. 13} ;
        end
        otherwise do
            {node does not require action} Add(currentNode) to visitedNodes;
        end
    end
    currentLink= getSuccessorLink(currentNode);
    currentNode= getTarget(currentLink) ;
    {check whether node is an EndPoint and stubs is an empty}
    if (currentNode == EndPoint and stubs is an empty) then
        Add(currentNode) to endPoints ;
        Add(currentNode) to visitedNodes;
    end
end
```

Figure 4.2. For example, applying CIA (i.e. modification change) with respect to $SC = (resp\_4, y)$. As shown in fig. 4.2(b), the impacted elements are colored in green, and the elements that are not impacted are colored in red. In addition, more explanation are illustrated in sect. 4.2.



(a) UCM Path Node with set of respRefs



(b) Impacted elements after applying CIA, $SC = (resp\_4, y)$

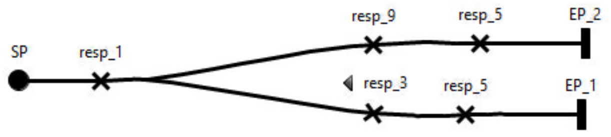
Figure 4.2: An example of handling respRefs

Algorithm 7: respRef algorithm

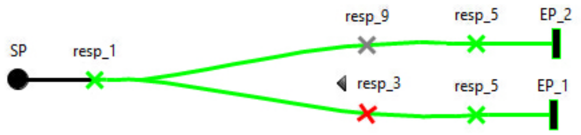
```
Procedure Name: HandlingResponsibility
Input  : currentRespRef:PathNode
Output: update the relevance of respRef whether it is relevant or irrelevant.
relevantRespRef:List(respRef);
irrelevantRespRef:List(respRef);
if ($currentRespRef \neq Construct(SC)$) then
    if ($currentRespRef$ not in $visitedNodes$) then
        add($visitedNodes$, $currentRespRef$);
    end
    {Check whether $currentRespRef$ is Relevant or Irrelevant
    True if $currentRespRef$ is Relevant}
    if ($DetermineConstructRelevancy(respRefExpression, SCVariables)$)
      then
        add($relevantRespRef$, $currentRespRef$);
        add($relevantNodes$, $currentRespRef$);
    else
        add($irrelevantRespRef$, $currentRespRef$);
    end
else
    {Either $currentRespRef$ selected as a SC, or Reaching Construct(SC),
      means a loop}
    call DetermineConstructRelevancy algorithm (see algo. 5)
end
```

4.3.2 Or-Fork's Procedure

When an Or-Fork is encountered during forward traversal, the outcoming links (i.e. successor branches) are traversed separately. For each branch, forward traversal will be executed reclusively. It accepts as input a branch link as a startLink and SCVariables list. For each branch, the forward traversal algorithm computes its own dependencies and specifies relevant/irrelevant nodes based on local dependency variables (i.e. SCVariables). In case of extracting internal or-Forks, each sub branches are handled separately in a recursive manner. Then, the computation of dependencies variables is computed according to the parent branch. The visitedJoins list defined in algorithm 6 is used in order to deduct the loop when exists. For example (see fig. 4.3), applying CIA (i.e. deletion change) with respect to SC = (*resp_1*, y). it shows the control flow paths colored in green, and relevant/irrelevant elements. Algorithm 8 describes the steps of handling the Or-Forks nodes.



(a) UCM Path Node with an Or-Fork



(b) Impacted elements after applying CIA, SC = (*resp_1*, y)

Figure 4.3: An example of handling OR-Fork

Algorithm 8: OR-Fork algorithm

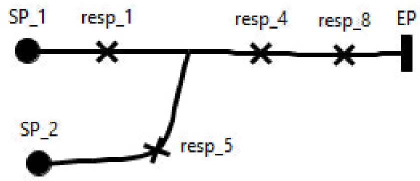
```
Procedure Name: HandlingOR-Fork
Input  : or-Fork:PathNode
Output: update SCVariables, visitedNodes, and visitedJoins lists
if (or − Fork ∉ visitedJoins then
    add(visitedJoins, or − Fork);
    add(visitedNodes, or − Fork);
    add(relevantNodes, or − Fork);
    foreach (Link l ∈ getSuccessor(or − Fork)) do
        result:List(String) = ForwardTraversal(l, SCVariables); {see algo. 6}
        add(SCVariables, result);
    end
else
    {Otherwise it's a loop} Exit_HandlingOF();
end
```

4.3.3 Or-Join's Procedure

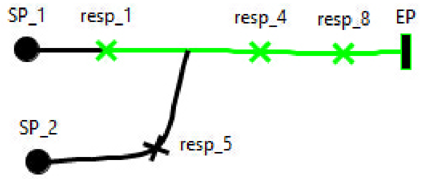
Or-Join construct receives at least one sub-branch (i.e. incoming lists). The algorithm 9 keeps tracking those incoming links in order to decide whether they will be impacted or not regarding control flow dependencies. Given an example (see fig. 4.4) of or-Join having two predecessors links within UCM model. To exercise Or-Join, applying CIA (i.e. addition change) with respect to *SC* = (*resp_1*, *y*). It shows the marked impacted elements with colored path nodes. As shown in fig. 4.4(b), just one incoming link is marked, as a result of the target criterion, that affected the control flow of execution.

4.3.4 Stub's Procedure

In order to improve that consistency among UCM model, we use stubs which are lower level of UCM sub-maps that hide the information in their plug-in (i.e. Plug-in Bindings). There are two types of plug-in Bindings (1) In-Binding binds the



(a) UCM Path Node with an Or-Join



(b) Impacted elements after applying CIA, $SC = (resp\_1,\ y)$

Figure 4.4: An example of handling Or-Join

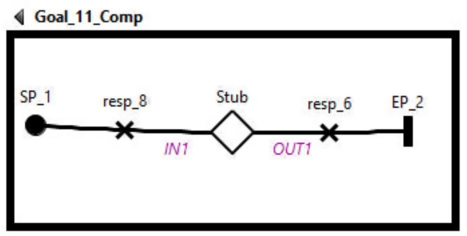
Algorithm 9: Or-Join algorithm

```
Procedure Name: HandlingOr-Join
Input  : or-Join:PathNode
Output: update SCVariables list
irrelevantBranches:List(NodeConnection) = $\emptyset$;
if $(or - Join = Construct(SC))$ then
    add($visitedJoins$, $or - Fork$);
    add($visitedNodes$, $or - Fork$);
    add($relevantNodes$, $or - Fork$);
    ForwardTraversal(getSuccessor($or - Join$), $SCVariables$); {see algo. 6}
else
    if $(or - Join \notin visitedNodes)$ then
        add($visitedNodes$, $or - Join$);
        foreach $(Link\ l \in getPredecessor(or - Join))$ do
            add($irrelevantBranches$, $l$);
        end
        remove($irrelevantBranches$, $currentLink$);
    else
        remove($irrelevantBranches$, $currentLink$);
    end
end
```

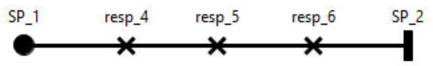
stubs In-Path to Start Point via a link in UCM map(s) (2) Out-Bindings binds the stubs Out-Path to End Point in UCM map(s). In addition to plug-in Bindings, there are three types of stubs which are static, dynamic, and synchronize. The dynamic stub may have more than one plug-in, and it requires traverse all plug-in in UCM mode. This traversing depends on the data flow dependency. To handle multiple stubs with their plug-in, we use stubs list to save the hierarchy level of stubs. In forward traversal, the stubs are handled in two steps (algorithm 10). (1) In-Binding allows to enter to plug-in maps, and (2) reaching to the End Point of this plug-in map when encountered End Point(s). then, continue traversing to parent maps using Out-Bindings (see Algo. 13). Figure 4.5 and Figures[4.5(c), 4.5(d)] shows a UCM model with multiple level of stubs and a UCM specification after applying a change impact analysis feature, respectively. For example, applying CIA (i.e. modification change) with respect to $SC = (resp\_8, y)$.

4.3.5 And-Fork's - And-Join's Procedures

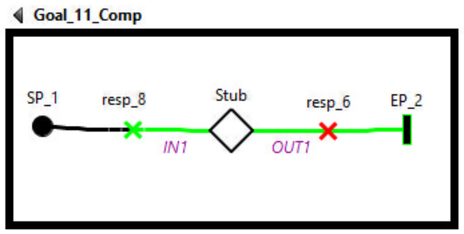
When and-Fork is encountered during the forward traversal. Since the and-Fork has at least two branches (see fig. 4.6(a)), so we need to perform forward traversal algorithm for each branch independently. In case and-Join is encountered, we move backward to catch all predecessor for the other branches of and-Join in order to perform forward traversal. For example, applying CIA (i.e. deletion change) with respect to $SC = (resp\_1, y)$.As shown in Fig. 4.6(b), the forward traversal is applied for both branch *EP_1* and *EP_2*.



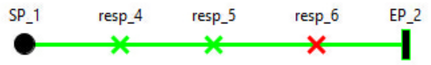
(a) UCM parent map



(b) Plugin map to *stub* in Parent map 4.5(a)



(c) Impacted elements after applying CIA



(d) Impacted elements after applying CIA

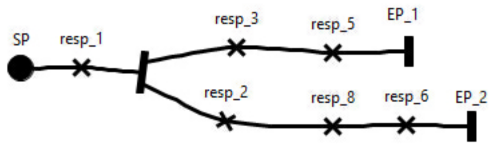
Figure 4.5: An example of handling Stub

Algorithm 10: Stub algorithm

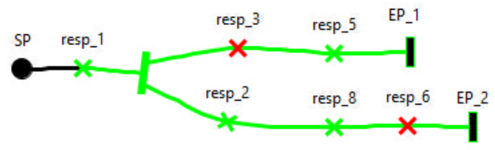
```
Procedure Name: HandlingStub
Input   : stub:PathNode
Output: update visitedNodes and SCVariables lists
irrelevantIN:List(NodeConnection) = $\emptyset$;
irrelevantOUT:List(NodeConnection) = $\emptyset$;
stubEntry:NodeConnection=$null$;
if ($stub \notin visitedNodes$) then
    add($visitedNodes$, $stub$);
    add($irrelevantIN$, predeccessorLink($stub$));
    add($irrelevantOUT$, SuccessorLink($stub$));
end
add($stub$, $stubs$);
foreach ($bindings{:}PluginBindings \in getBindings(stub)$) do
    foreach ($IN : InBinding \in getInBindings(stub)$) do
        if ($entryLinkOf(IN) == currentLink$) then
            $stubEntry$ = getStartPointOf($IN$);
            add($visitedNodes$, $stubEntry$);
        end
        result:List(String) = ForwardTraversal(SuccessorLinkOF($stubEntry$),
         $SCVariables$); {see algo. 6}
        add($SCVariables$, $result$);
    end
end
```

Algorithm 11: And-Fork algorithm

```
Procedure Name: HandlingAnd-Fork
Input   : and-Fork:PathNode
Output: update Groups, visitedNodes lists
if $(and - Fork \notin visitedJoins)$ then
    add($visitedNodes$, $and - Fork$);
    foreach $(Link\ l \in getSuccessorLinks(and - Fork))$ do
        ciaForward = ForwardTraversal($l$, $SCVariables$); {see algo. 6}
          add($group$, $ciaForward$);
    end
    add($Groups$, $group$);
else
    {If and-Fork contained in VisitedNodes, means a loop}
end
```



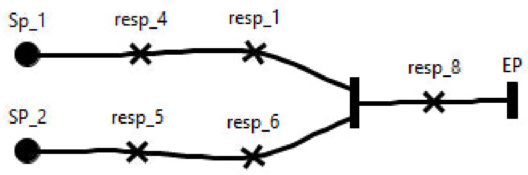
(a) UCM Path Node with a And-Fork



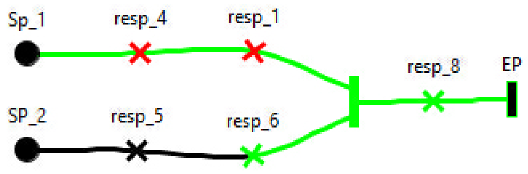
(b) Impacted elements after applying CIA, $SC = (resp\_1,\ y)$

Figure 4.6: An example of handling And-Fork

Another case when SC resides within concurrency branch (i.e. enclosed between And-Fork and And-Join), first, before starting the forward traversal, the CIA algorithm move backward to get all the other concurrent branches in order to perform forward traversal for each branch independently. For example, applying CIA (i.e. addition change) with respect to SC = ($resp\_6$, -) (see Fig. 4.7(a)). As a result of CIA if Fig. 4.7(b), the forward traversal is applied normally to the branch that the SC resides in and also the forward traversal is applied to the rest of concurrent branch after moving backward to get them.



(a) UCM Path Node with an And-Join



(b) Impacted elements after applying CIA,
SC = ($resp\_6$, t)

Figure 4.7: An example of handling And-Join

In case of and-Fork and and-Join, both require a different procedure than or-Fork and or-Join procedure, because the concurrency enclosed between them, so the order of execution is an issue that should be considered during a forward traversal of the sequences of paths (i.e. concurrent branches). An or-Fork is a path node that exclusively executes on branches (i.e. always all paths) whereas and-Fork allows paths to execute in concurrency manner, but and-Join receives at least two predecessor branches, on the other word, it waits for all incoming paths.

```
Algorithm 12: And-Join algorithm
Procedure Name: HandlingAnd-Join
Input  : and-Join:PathNode
Output: Groups:List(ciaForwardAlgorithm)
backwardLinks:List(NodeConnection) = $\emptyset$;
if ($and - Join = Construct(SC)$) then
    add($visitedJoins$, $and - Fork$);
    add($visitedNodes$, $and - Fork$);
    add($relevantNodes$, $and - Fork$);
    ForwardTraversal(getSuccessor($and - Join$), $SCVariables$); {see algo. 6}
else
    if ($and - Join \notin visitedJoins$) then
        add($visitedNodes$, $and - Join$);
        foreach ($Link$ l $\in$ $getPredecessorLinks(and - Join)$) do
            if (l $\neq$ $cuurentLink$) then
                add($backwardLinks$, getStartLink($l$));
            end
        end
        foreach ($Link$ l $\in$ $backwardLinks$) do
            ciaForwardAlg:forwardAlgrithm = new instance of ciaForwardAlgo.
            ciaForwardAlg = (call ForwardTraversal($l$,SCVariables);
              add($Groups$, ciaForwardAlg);
        end
    end
end
```

4.3.6 EndPoint's Procedure

An End Point construct is same (i.e. respecting to control flow) as Or-Fork path nodes. It may determine the execution of the path within another plug-in. when encountering End Point, that does not mean that we reach to the end of forward traversal unless there is no stub left un-traversed branches/paths. Algorithm 13 describes the steps of handling the End Point.

Algorithm 13: EndPoint algorithm

```
Procedure Name: HandlingEndPoint
Input  : endPoint:PathNode
Output: update SCVariables
irrelevantOUT:List(NodeConnection) = $\emptyset$;
stubExit:NodeConnection = null;
if (endPoint $\notin$ visitedNodes) then
    add(visitedNodes, endPoint);
end
if (stub not empty) then
    foreach (PluginBinding : binding $\in$ getBindings(stub)) do
        foreach (OutBindingOutB $\in$ getOut(stub)) do
            if (endPointOf(OutB) == endPoint) then
                stubExit = getStubExitOf(OutB);
                remove(irrelevantOUT, stubExit);
                result:List(String) = ForwardTraversal(stubExit,
                  SCVariables); {see algo. 6}
                add(SCVariables, result);
            end
        end
    end
end
```

4.4 Change impact through URN links

This step aims at identifying other potential URN impacted elements by following existing URN Links. A URN Link is used to create a connection between any two URN elements, e.g., intentional element reference/definition, actor reference/definition, link, responsibility reference/definition, component reference/definition, etc. URN links may be used to represent traceability information between different URN elements, e.g., a GRL task is implemented using a UCM responsibility. This would allow for consistency analysis [56].

We define URN links as follows:

Definition 4.6 (URN Link) *A URN Link is defined as urnl = (type, from, to),* *where* type *denotes a user-defined URN Link type,* from *denotes the ID of the source URN element, and* to *denotes the ID of the target URN element.*

Algorithm 14 iterates through the set of impacted elements (i.e., *relevantURN-Constructs*) and checks whether these elements are involved in any URN Link, as source (i.e., *from* field) or as a target (i.e., *to* field). Note that the relevantURN-Constructs list contains all impacted elements (i.e. relevant URN constructs - UCM and GRL). Since an impacted element can serve as a source or a target in a URN Link and since one source element can be linked to many target elements and vice versa, we have used two search functions to retrieve the set of elements IDs depending whether we are looking for source or target IDs. (i.e., search-SourceURNLinks and searchTargetURNLinks). The new identified elements are then add to the set *relevantURNConstructs* list.

Algorithm 14: Excerpt of the algorithm to identify impacted elements emanating from URN Links

```
Procedure Name: IdentificationOfOverallImpactedElements
Input   : URN Model + relevantNodes
Output: relevantURNConstructs
URNLinksList = getAllURNLinks();
relevantURNConstructs:List = relevantNodes;
foreach (e ∈ relevantURNConstructs) do
    {Search for target elements IDs when e is defined as source};
    ToElementList = searchTargetURNLinks(e,from,URNLinksList);
    AddToRelevantURNConstructs(ToElement, relevantURNConstructs);
    {Search for source elements IDs when e is defined as target}
     FromElementList = searchSourceURNLinks(e, URNLinksList);
    AddToRelevantURNConstructs(FromElement, relevantURNConstructs);
end
```

4.5 jUCMNav UCM-based Change Impact Analysis Feature

Our proposed change impact analysis approach is implemented as a feature [1]2]The CIA feature is publicly available and can be downloaded from `https://github.com/JUCMNAV/projetseg`. within the jUCMNav framework [13], a full graphical editor and analysis tool for UCM models developed as an Eclipse-based plug-in.

To exercise this feature, the user starts by selecting an UCM construct. Then then right-clicks to choose from sub-menu commands: *Addition*, *Deletion*, or *Modification* (see Fig. 4.8). For the addition and modification option, it is required that the analyst adds the UCM construct first then call the feature as required.

Three types of changes (Addition, Modification, and Deletion) are supported on responsibilities, or-Fork branches, or-Join, and-Fork branches, and-Join, Timer , or StartPoint, where the user can target specific variables, if any. It is worth noting that deletion do not require variables selection.

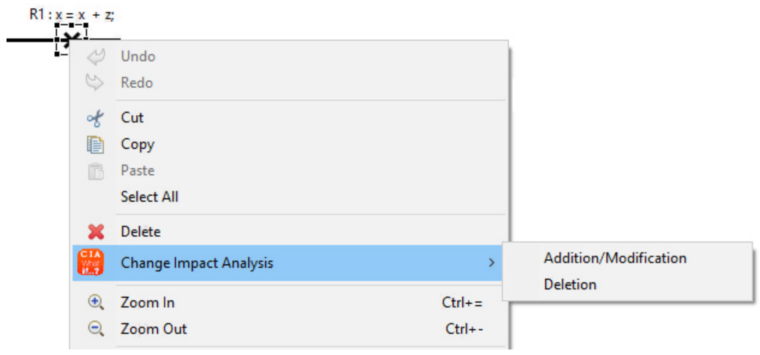
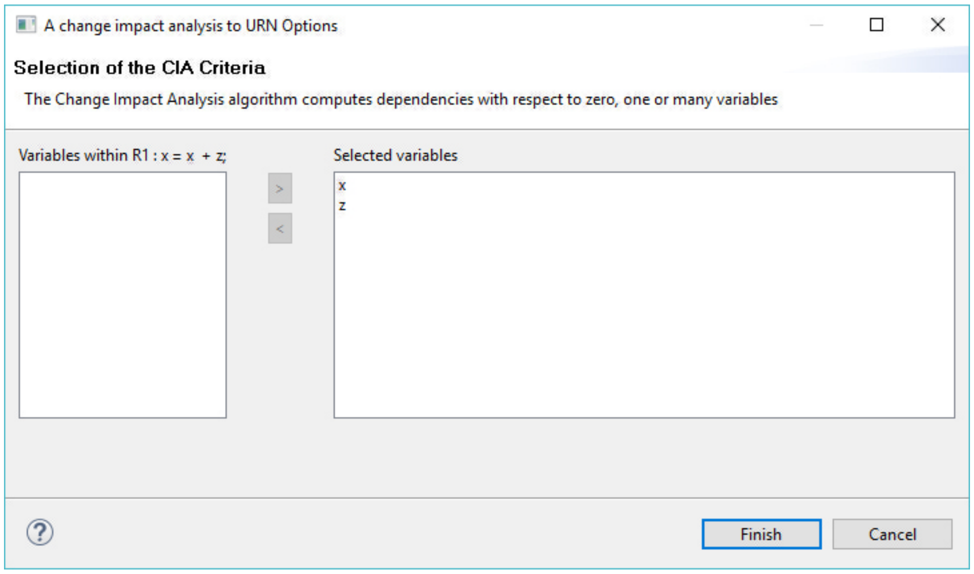


Figure 4.8: UCM CIA included in command menu of jUCMNav framework

In case of addition and modification, this type of change requires variable selection criteria. The selection criteria (i.e. code expressions) window will appear. (see Fig. 4.9). In the *left* box, all variables reside in the chosen criterion are listed. Then, the analyst may select these variables from left box to move them to *right* box Selected variables by using the arrow button. Note the analyst may be select zero or many variables as slicing criterion variables in order to identify the impact of change in the existing model with respect to these criterion variables. if the user does not select any variable, the CIA algorithm will treat with construct as an empty code.

Also, in case the responsibility or Or-Fork branch do not have code, the CIA algorithm will execute without computing the data flow dependencies between elements and just will do control flow dependencies. the result of this case is colored all elements come after selected element even if they do not have code or irrelevant elements to slicing criteria. In addition to the colored element, also, paths are colored with green color. Note that the impacted elements are marked in green color, the elements that are not impacted are colored in red color, and the elements that do not have any embedded code are colored in gray color.

The closure CIA approach marks all relevant UCM elements of the original model with respect to slicing criterion variables, means temporary coloring. Also, it cannot be saved color because the nature of jUCMNav tool does not support this feature. This approach helps the analyst to observe which parts of the model are impacted by modification. Moreover, it reduces the time consuming to figure out the impact analysis and avoid error activity in large and complex URN model. If


A change impact analysis to URN Options

Selection of the CIA Criteria

The Change Impact Analysis algorithm computes dependencies with respect to zero, one or many variables

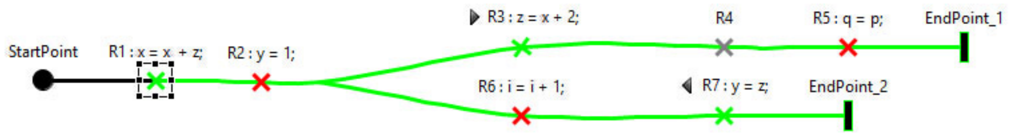
Variables within R1 : x = x + z;

Selected variables

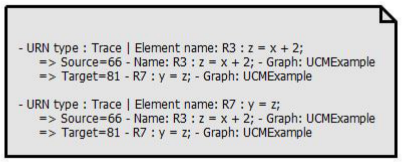
x
z

Figure 4.9: Selection criterion window in jUCMNav framework

any of the impacted element (marked in UCM model (see Fig. 4.10(a))), is part of UCM model, linked to any other UCM constructs through URN links, the details about impacted URN links, such as SourceID, TargetID, and Type, will appear as a UCM comment as shown in (see Fig.4.10(b)).



(a) UCM constructs impacted elements


- URN type : Trace | Element name: R3 : z = x + 2;
    => Source=66 - Name: R3 : z = x + 2; - Graph: UCMExample
    => Target=81 - R7 : y = z; - Graph: UCMExample

- URN type : Trace | Element name: R7 : y = z;
    => Source=66 - Name: R3 : z = x + 2; - Graph: UCMExample
    => Target=81 - R7 : y = z; - Graph: UCMExample

(b) Information about impacted URN links

Figure 4.10: Identification of impacted elements in UCM model

CHAPTER 5

EXPERIMENTAL EVALUATION AND VALIDATION

5.1 URN Experimental Evaluation

In this section, we evaluate our proposed change impact analysis approach using one mock model and three real-world GRL case studies of different sizes, complexity, and features. Table 5.1 provides some characteristics of the used case studies in terms of number of URN models (i.e. representing root maps/Graph and plugins), number of GRL Elements (Intentional Elements and Indicators), number of GRL links (i.e. connecting two GRL elements), number of UCM constructs (respRefs, OR-Forks, AND-Forks, etc.), number of Actors/Components (i.e. Actors are describing its intentions and capabilities / Components are characterized by its kind (Team, process, agent, etc.)), and number of URN links which are representing the traceability link between URN elements.

URN Spec.	Nb. of Models	Nb. of GRL Elements	Nb. of GRL Links	Nb. of UCM Constructs	Nb. of Actors / Components	Nb. of URN Links
Mock Model	18	66	71	22	19	7
Adverse Event Management System (AEMS)	7	29	13	15	5	9
Commuting System	11	19	20	18	4	8
Patient Discharge Process	56	61	64	44	13	12

Table 5.1: Case studies characteristics

5.1.1 Mock System

In order to cover all URN constructs, we have created a mock system that has many models, many actors, many components, all types of GRL elements and UCM constructs, all types of GRL links, many URN links of different types. We created a mock model to prove the effectiveness and accuracy of our proposed work to analyze the impact change of URN model.

(A) GRL Model

Figure 5.1illustrates part of GRL model constituting the mock model.

As a result of applying CIA based on different slicing criterion with different graphs within model. The test cases listed as follow: SC = (Element, Change type).

- *SC* = (*Task_6* in Graph (see fig. 5.1), *Addition*, -)
- *SC* = (*Task_5* in Graph (see fig. 5.1), *Deletion*, -)

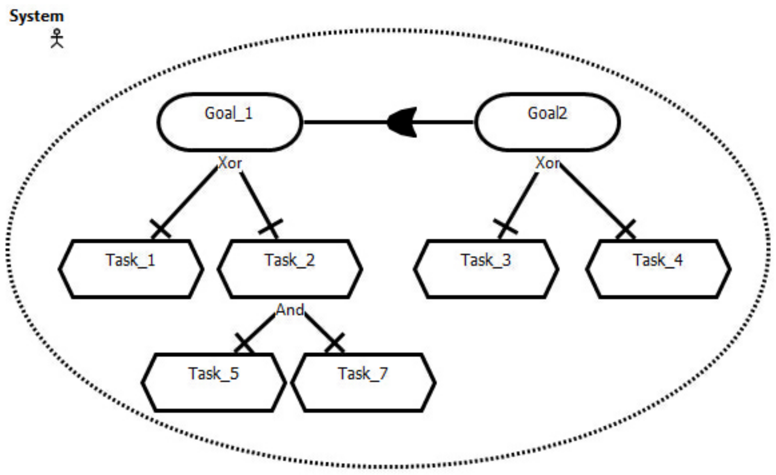


Figure 5.1: Mock system - GRL

- *SC* = (*DependencyLink* in Graph (see fig. 5.1), *Modification*, -)

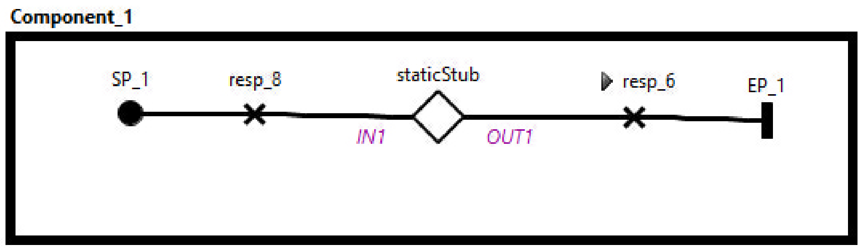
The test cases above, exercising GRL elements and Links and how handles each one of them based on proposed algorithms. The result of applying CIA to these cases are illustrated in appendix A.

(B) UCM Model

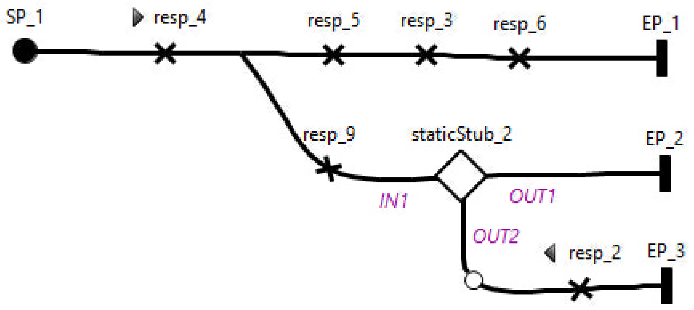
Figure 5.2illustrates part of UCM models constituting the mock model.

As a result of applying CIA based on different slicing criterion with different maps within model. The test cases listed as follow: SC = (*Element*, *Change type*, *SC Variables*).

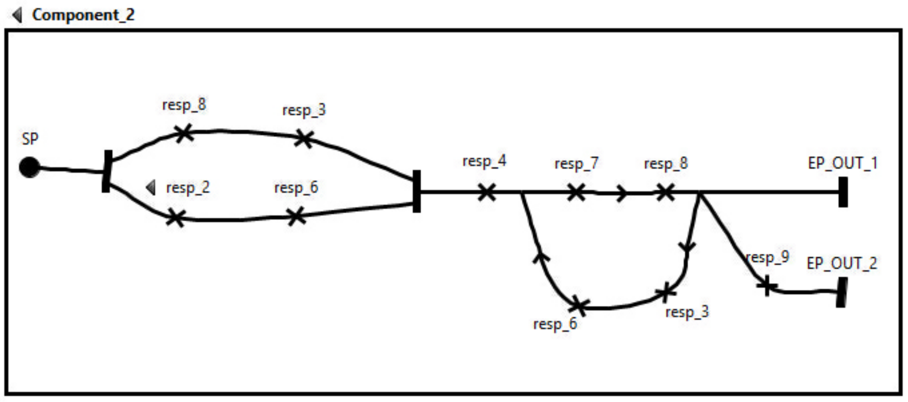
- *SC* = (*resp_8* in MainMap (see fig. 5.2(a)), *Modification*, *y*)
- *SC* = (*resp_9* in staticStubMap (see fig. 5.2(b)), *Addition*, -)
- *SC* = (*resp_4* in staticStub_2 (see fig. 5.2(c)), *Modification*, *(z,y)*)
- *SC* = (*resp_4* in staticStub_2 (see fig. 5.2(c)), *Deletion*, *z*)



(a) "Mock model - Main map UCM"



(b) "staticStub" plug-in Map for static stub *staticStub*



(c) "staticStub_2" plug-in Map for static stub *staticStub_2* UCM

Figure 5.2: Mock UCM model

- *SC* = (*resp_8* in staticStub_2 (see fig. 5.2(c)), *Deletion*, *t,n*)
- *SC* = (*resp_3* in staticStub_2 (see fig. 5.2(c)), *Addition*, *t*)
- *SC* = (*resp_4* in staticStubMap (see fig. 5.2(b)), *Modification*, *(z,y)*)

The test cases above, exercising URN constructs and how handles each one of them based on proposed algorithms. The result of applying CIA to these cases are illustrated in appendix A.

5.1.2 Cases Studies

In addition of the Mock model, we also implemented the CIA approach on three publicly available case studies that vary in size and complexity, as shown in Table 5.1.

Case Study 1: Adverse Event Management System (AEMS)

This case study describes an adverse event management system (AEMS) for a hospital.

(A) GRL Model

Figure 5.3 illustrates one of GRL models constituting the case study.

The first CIA task aims to identify potential impacted elements if we modify softgoal *FastProcess* (i.e., the GMDG node corresponding to *FastProcess* is used as slicing criterion to execute Algorithm 2). The produced GMDG is shown in Fig. 5.4, while the impacted GRL elements are shown in Fig. 5.5. Since the goal *comply with Privacy Laws* is only linked to the rest of the model through a URN

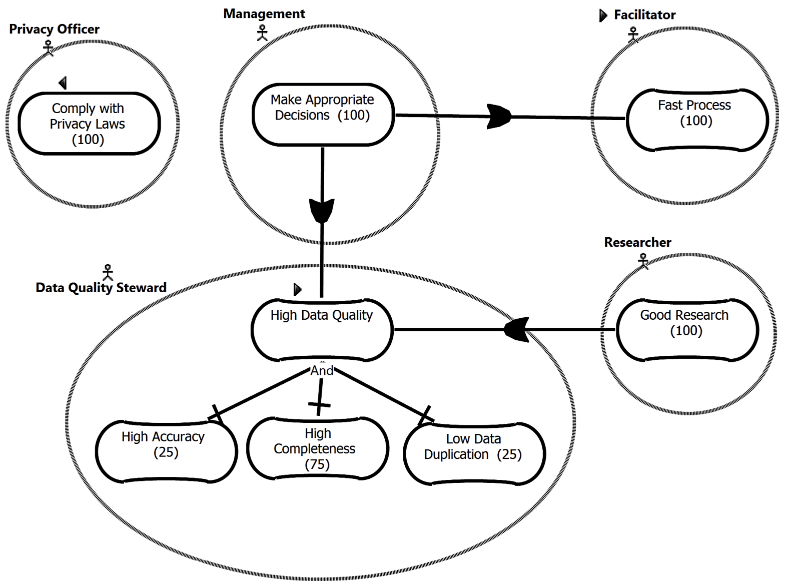


Figure 5.3: AEMS GRL Model

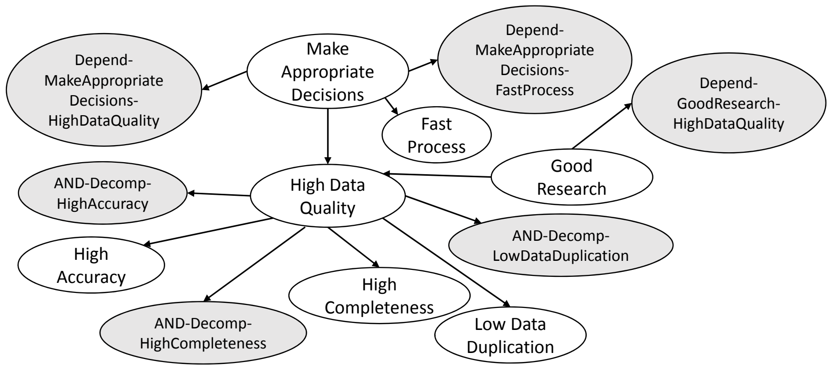

Depend-MakeAppropriateDecisions-HighDataQuality
Make Appropriate Decisions
Depend-MakeAppropriateDecisions-FastProcess
Fast Process
Depend-GoodResearch-HighDataQuality
Good Research
AND-Decomp-HighAccuracy
High Data Quality
AND-Decomp-LowDataDuplication
High Accuracy
High Completeness
AND-Decomp-HighCompleteness
Low Data Duplication

Figure 5.4: GMDG Graph corresponding to the AEMS GRL model of Fig. 5.3

Link, called *trace* (having its source at softgoal *High Data Quality*), there is no GMDG node associated with it.

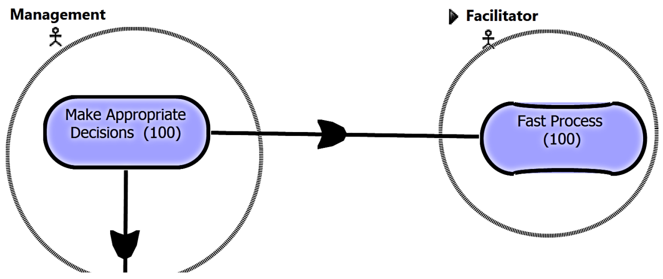


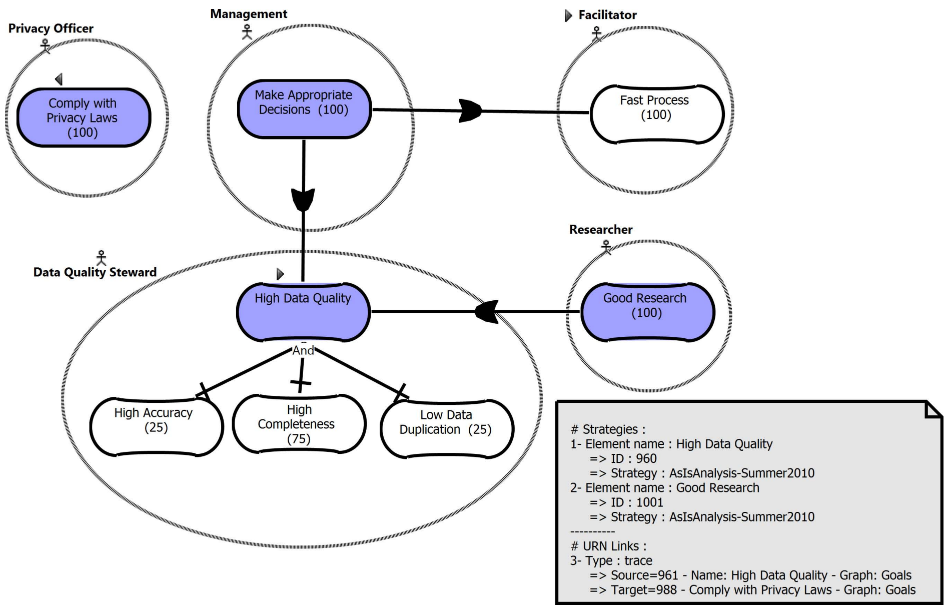
Figure 5.5: Impacted elements of the first AEMS CIA task

The second CIA task aims to identify potential impacted elements once we modify the softgoal *High Data Quality*. Three elements are impacted (i.e., goal *Make Appropriate Decisions*, and softgoals *High Data Quality* and *Good Research*) as a result of slicing the GMDG graph with the GMDG node that corresponds to *High Data Quality* as slicing criterion. In addition, goal *Comply with Privacy Law* is impacted since it is the target of the URN Link *trace*, having its source at softgoal *High Data Quality*. Finally, one evaluation strategy is identified, called *AsIsAnalysis-Summer2010*, involving both softgoals *High Data Quality* and *Good Research*. Figure 5.6 illustrates the impacted elements.

(B) UCM Model

Figure 5.7 illustrates two UCM models constituting the case study.

The CIA task aims to identify potential impacted elements if we modify responsibility *WarnObserver* (i.e. is used as slicing criterion with its variable to execute in algorithm 6 - *SC* =(WarnObserver, *EventReady*)). The impacted UCM elements are shown in Fig. 5.7 itself. In addition, responsibility *WarnObserver* is



```
# Strategies :
1- Element name : High Data Quality
    => ID : 960
    => Strategy : AsIsAnalysis-Summer2010
2- Element name : Good Research
    => ID : 1001
    => Strategy : AsIsAnalysis-Summer2010
----------
# URN Links :
3- Type : trace
    => Source=961 - Name: High Data Quality - Graph: Goals
    => Target=988 - Comply with Privacy Laws - Graph: Goals
```

Figure 5.6: Impacted elements of the second AEMS CIA task

impacted since it is the target of the URN Link *Observer*, having its source at intentional element *Number of events returned to Observers.* Figure 5.7(c) shows the details of URN Links, Consisting of the name of the element, source, target, and map name.

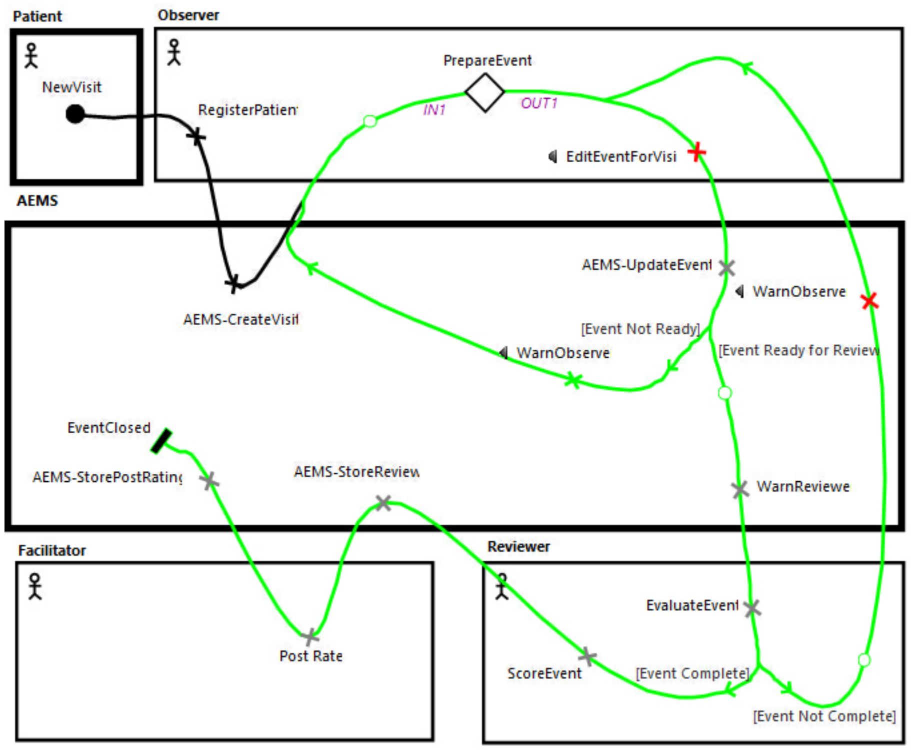
Table 5.2 listed all responsibilities and or-Forks that have code expressions. The URN Links within the model and Plug-in bindings of the stubs are listed in Tables 5.4 and 5.3, respectively.

Case Study 2: Commuting System

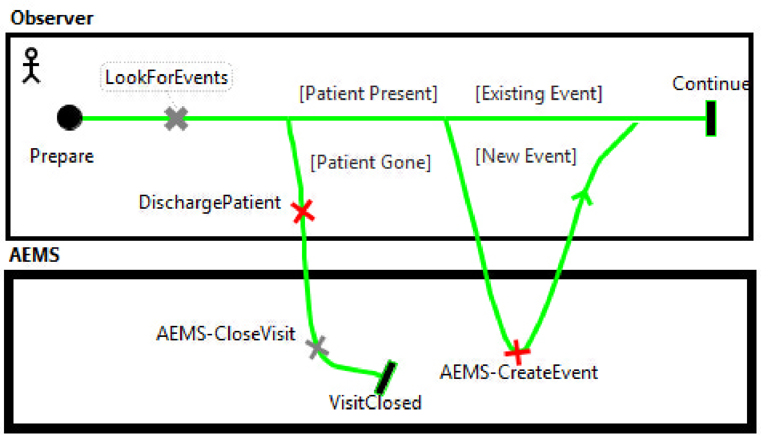
The second case study is a URN model specification describing a commuting system.

(A) GRL Model

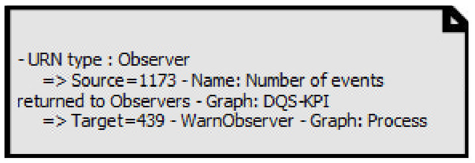
The second case study is a GRL specification describing a commuting system



(a) "Process" Map UCM



(b) "Prepare Event" plug-in Map UCM



- URN type : Observer
  => Source=1173 - Name: Number of events returned to Observers - Graph: DQS-KPI
  => Target=439 - WarnObserver - Graph: Process

(c) Identified URN Links within AEMS model

Figure 5.7: AEMS UCM Model

Element Name	Type	Map	Expression / Condition
NewVisit	StartPoint	Process	-
RegisterPatient	RespRef	Process	EventsCreated = 0; ExistingEvent = false; Discharged = false;
AEMS-CreateVisit	RespRef	Process	-
EditEventForVisit	RespRef	Process	ExistingEvent = false;
AEMS-UpdateEvent	RespRef	Process	-
WarnObserver	RespRef	Process	ExistingEvent = true; EventComplete = true; EventReady = true;
WarnObserver	RespRef	Process	ExistingEvent = true; EventComplete = true; EventReady = true;
WarnReviewer	RespRef	Process	-
EvaluateEvent	RespRef	Process	-
ScoreEvent	RespRef	Process	-
AEMS-StoreReview	RespRef	Process	-
Post Rate	RespRef	Process	-
AEMS-StorePostRating	RespRef	Process	-
Event Not Ready	Or-Fork branch	Process	!EventReady
Event Ready for Review	Or-Fork branch	Process	EventReady
Event Complete	Or-Fork branch	Process	EventComplete
Event Not Complete	Or-Fork branch	Process	!EventComplete
LookForEvents	RespRef	PrepareEvent	-
DischargePatient	RespRef	PrepareEvent	Discharged = true;
AEMS-CloseVisit	RespRef	PrepareEvent	-
AEMS-CreateEvent	RespRef	PrepareEvent	EventsCreated = EventsCreated + 1;
Patient Present	Or-Fork branch	PrepareEvent	(NumEvents>EventsCreated) || ExistingEvent
Patient Gone	Or-Fork branch	PrepareEvent	else
New Event	Or-Fork branch	PrepareEvent	(EventsCreated<NumEvents) && !ExistingEvent
Existing Event	Or-Fork branch	PrepareEvent	else

Table 5.2: Adverse Event Management System (AEMS) model information

Stub name	Plug-in map	IN binding	OUT binding
PrepareEvent	Process	IN1 <–>Prepare	OUT1 <–>Continue

Table 5.3: Plug-in bindings of stubs in AEMS model

URN Link Type	**ElementName as Source**	**src Map**	**ElementName as Target**	**trgt Map**
Trace	(GRL) High Data Quality	Goals	(GRL) Comply with Privacy Laws	Goals
Trace	(UCM) Edit EventForVisit	Process	(GRL) Number of events with patient information	Privacy-KPI
-	(GRL) Number of events	DQS-KPI	(UCM) WarnObserver	Process

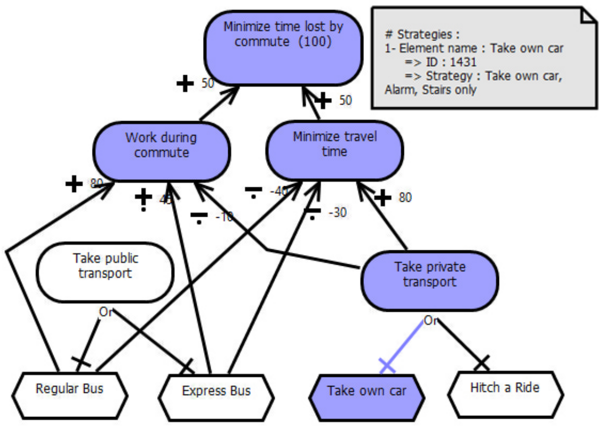
Table 5.4: URN links in AEMS Model

which is consists of 4 specifications, 22 Intentional Elements, and 10 URN links. Figure 5.8 shows the impact (in purple) of changing the task *Take own car*, on both models Commuting-Time (Fig. 5.8(a)) and Stakeholders (Fig. 5.8(b)). The impacted elements are part of a strategy, called *Take own car, Alarm, Stairs only.*

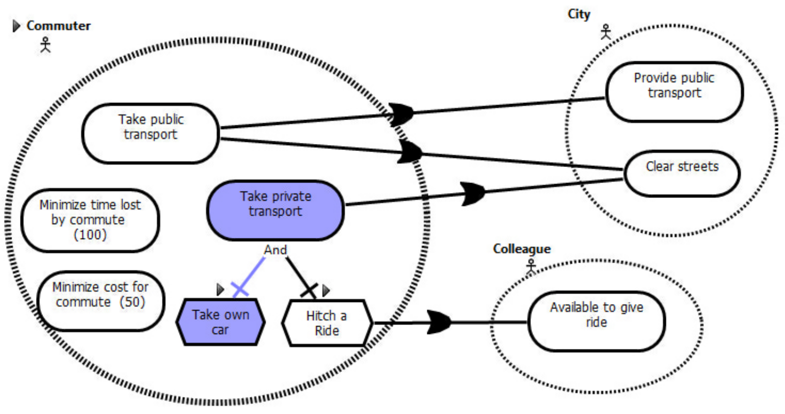
(B) UCM Model

The second case study is a UCM model specification describing a commuting system. The first CIA task aims to identify the potentially impacted elements once we delete the start point *ready to leave home.* As a result of this modifying, the entire scenario will be impacted, and the stub *arm system* (i.e. Alram System plug-in stub) will be impacted with all its component. Since there is no any URN link established between impacted elements, the URN links details will not be shown. Figures 5.9(b) and 5.9(c) illustrate the impacted elements within Commuting system itself.

The second CIA task aims to identify the potential impacted elements if we modify responsibility (i.e. *take #100*) (see Fig. 5.10(c)) which resides between



(a) Impacted elements in the Commuting-Time Model



(b) Impacted elements in the Stakeholders Model

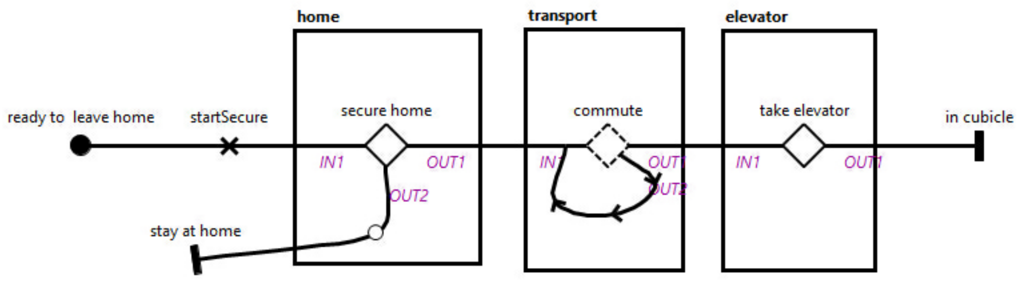
Figure 5.8: Identification of impacted elements in two GRL models of the commuting case study

or-Join and and-Join. The impacted elements are shown in Fig.(see Fig. 5.10(c)) itself.

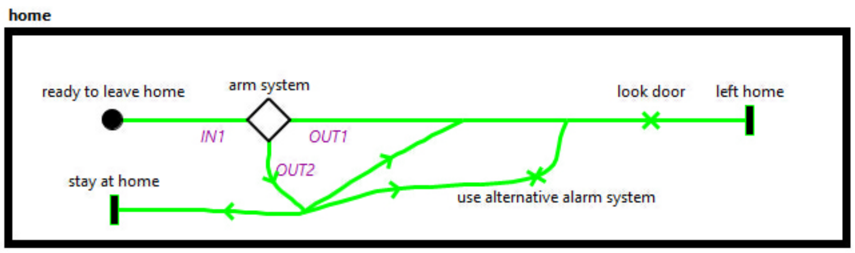
Table 5.5 listed all responsibilities and or-Forks that have code expressions. The URN Links within the model and Plug-in bindings of the stubs are listed in Tables 5.7 and 5.6, respectively.

Elements Name	Type	Map	Expression / Condition
look door	RespRef	Secure Home	ReadyToLeft = true
use alternative alarm system	RespRef	Secure Home	-
noAlarmChoice Unsecured	Or-Fork branch	Secure Home	NoAlarmChoice = UNSECURED
noAlarmChoice Alternate	Or-Fork branch	Secure Home	NoAlarmChoice = ALTERNATE
noAlarmChoice Home	Or-Fork branch	Secure Home	NoAlarmChoice = HOME
accept code	RespRef	Arm System	-
check code	RespRef	Arm System	CodeChecked =true
notArmed	Or-Fork branch	Arm System	QuitAlarm && CodeChecked
matched	Or-Fork branch	Arm System	Matched
not matched	Or-Fork branch	Arm System	!Matched
drive car	RespRef	Car	-
break down	RespRef	Car	GRL_Take_own _car = 0
problem	Or-Fork branch	Car	CarProblem
no problem	Or-Fork branch	Car	!CarProblem
hitch a ride in car	RespRef	Hitch a Ride	-
deal with work email	RespRef	Regular Bus	-
take #95	RespRef	Regular Bus	-
take #97	RespRef	Regular Bus	-
take #96	RespRef	Regular Bus	-
BusChoice	Or-Fork branch	Regular Bus	BusChoice = Number95
BusChoice	Or-Fork branch	Regular Bus	BusChoice = Number97
deal with work email	RespRef	Express Bus	-
take #100	RespRef	Express Bus	-
call elevator	RespRef	Take Elevator	-
select floor	RespRef	Take Elevator	-
take stairs	RespRef	Take Elevator	-

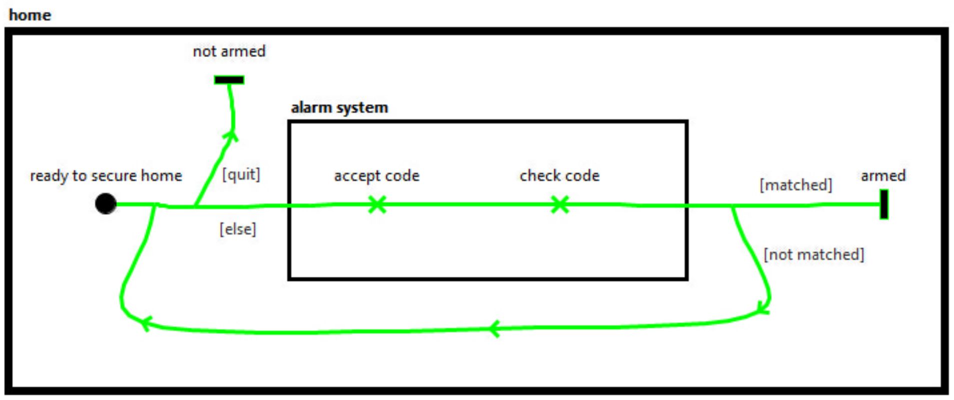
Table 5.5: Commuting model information



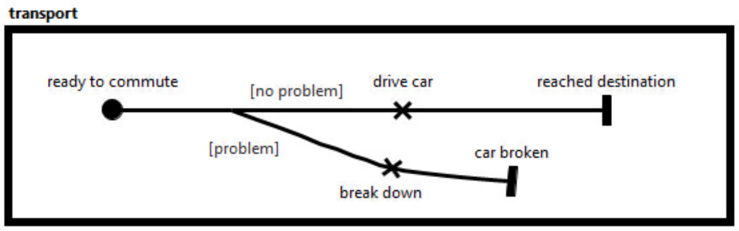
(a) Commuting map



(b) Secure Home map

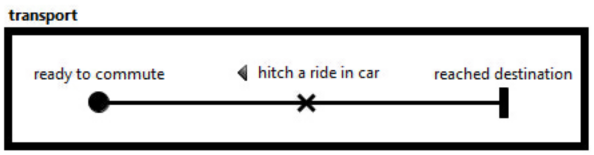


(c) Alrm System map

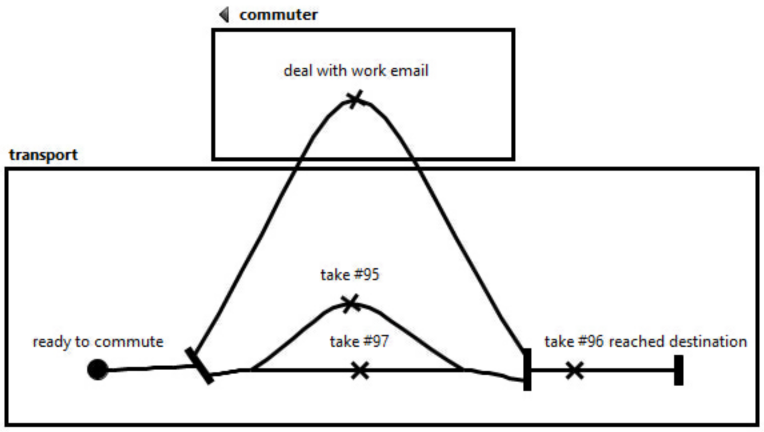


(d) Car map

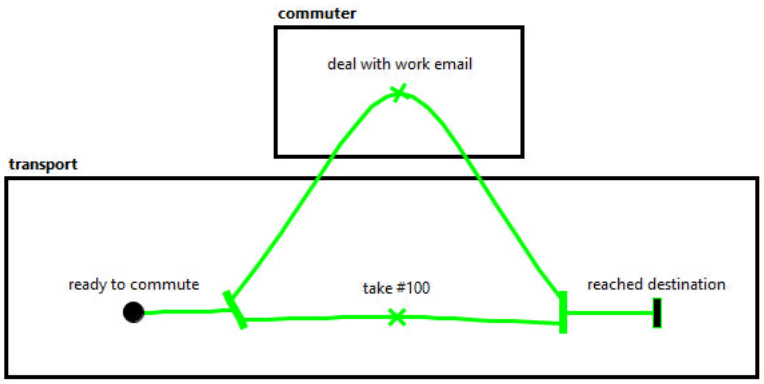
Figure 5.9: Commuting model - Part 1



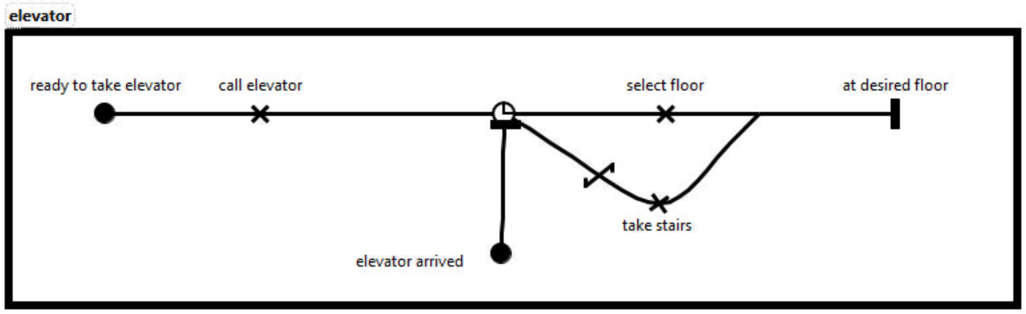
(a) Hitch a Ride map



(b) Reqular Bus map



(c) Express Bus map



(d) Take Elevator map

Figure 5.10: Commuting model - Part 2

Stub name	**Plug-in map**	**IN bindings**	**OUT bindings**

secure home	Commuting	IN1<->ready to leave home	OUT1<->left home
			OUT2<->stay at home

commute	Car	IN1<->ready to commute	OUT1<->reach destination OUT2<–>car broken
commute	Hitch a Ride	IN1<->ready to commute	OUT1<->reach destination
commute	Regular Bus	IN1<->ready to commute	OUT1<->reach destination
commute	Express Bus	IN1<->ready to commute	OUT1<->reach destination
take elevator	Take Elevator	IN1<->ready to take elevator	OUT1<->at desired floor

Arm system	Arm System	IN1<->ready to secure home	OUT1 <->armed
			OUT2 <->not armed

Table 5.6: Commuting model information

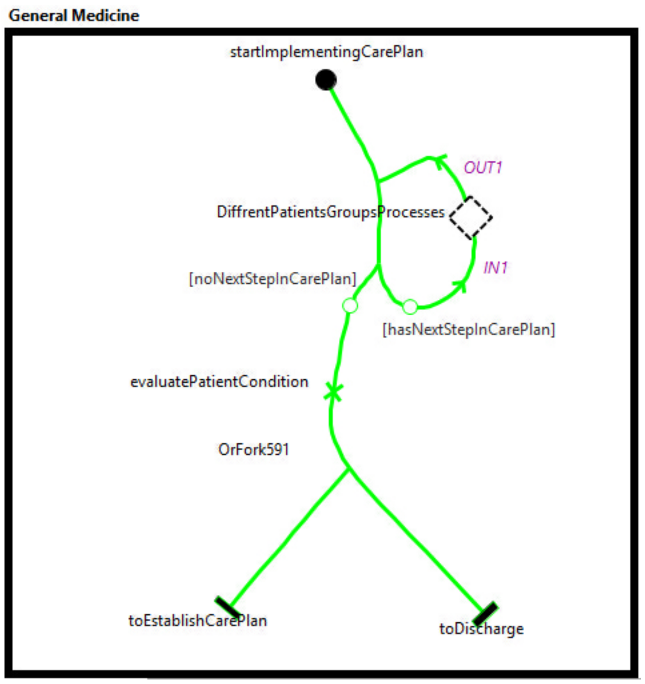
URN Link Type	**ElementName as Source**	**src Map**	**ElementName as Target**	**trgt Map**
Trace	(GRL) Take own car	Stakeholders	(UCM) drive car	Car
Trace	(GRL) Hitch a Ride	Stakeholders	(UCM) hitch a ride in car	Hitch a Ride
-	(GRL) Commuter	Stakeholders	(UCM) commuter	Regular Bus

Table 5.7: Commuting model information

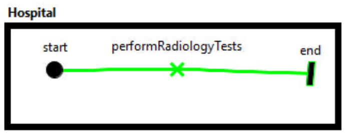
Case Study 3: Patient Discharge Process

This last public URN model describes the patient discharge process at The Ottawa Hospital. Due to the large size of the URN specification (which contains 56 maps) and because of the lack of space, the reader is referred to [57]to consult the original URN model. The main reason why this model was selected is to test the scalability of the approach to large models.

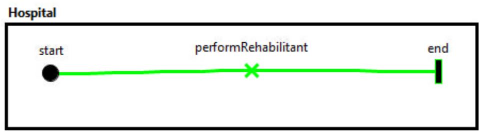
The CIA is applied on the model. The selected element as a SC, *(*startImplementingCarePlan), resides within *(*CarePlanImplementation map). The map contains one dynamic stub, so the forward algorithm will traverse all the contained stubs as shown in Fig. 5.11.



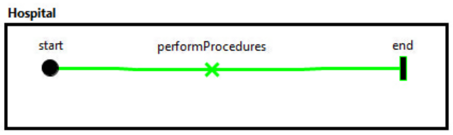
(a) Care Plan Implementation map



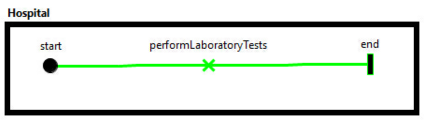
(b) Radiology Tests map



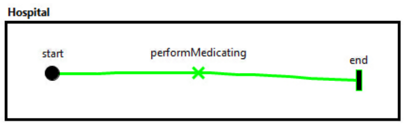
(c) Rehabilitant map



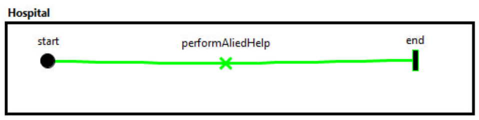
(d) Procedures map



(e) Laboratory Tests map



(f) Medicating map



(g) Allied Help map

Figure 5.11: Patient Discharge Process

5.2 Experimental Validation

Our main goal of this experiment is to check whether the use of change impact analysis feature improves the comprehension of URN model. In what follows, we formulated research question in order to seek for the answers.

The experiment investigates the following question:

Is the use of Change Impact Analysis feature would identify precisely the URN impacted elements with respect to a maintenance task?.

5.2.1 Experiment planning

The main goal of conducting this empirical study is to analyze that change impact analysis feature is identifying precisely the URN impacted elements with respect to a maintenance task. We design and conduct an experiment in order to test the derived hypothesis (see Sect. 5.2.6).

Kitchenham et al. [58], Jedlitschka and Ciolkowski [59], Wohlin et al. [60], and Juristo and Moreno [61], introduced the guidelines, recommendation, and templates in order to show the analysis and statistics. Figure[5.12] illustrates an overview of the experimental plan of conducting experiments. Next, we explain each step of an experiment in detail. As shown in fig. 5.12, our experiment is based on the data collected from the subjects of by using (1) Descriptive analysis (2) Independent-Samples t-test to analyze the data. In addition, we asked experts to manually identify the change impact analysis for the models in order to validate the accuracy of approach by computing the precession and recall.

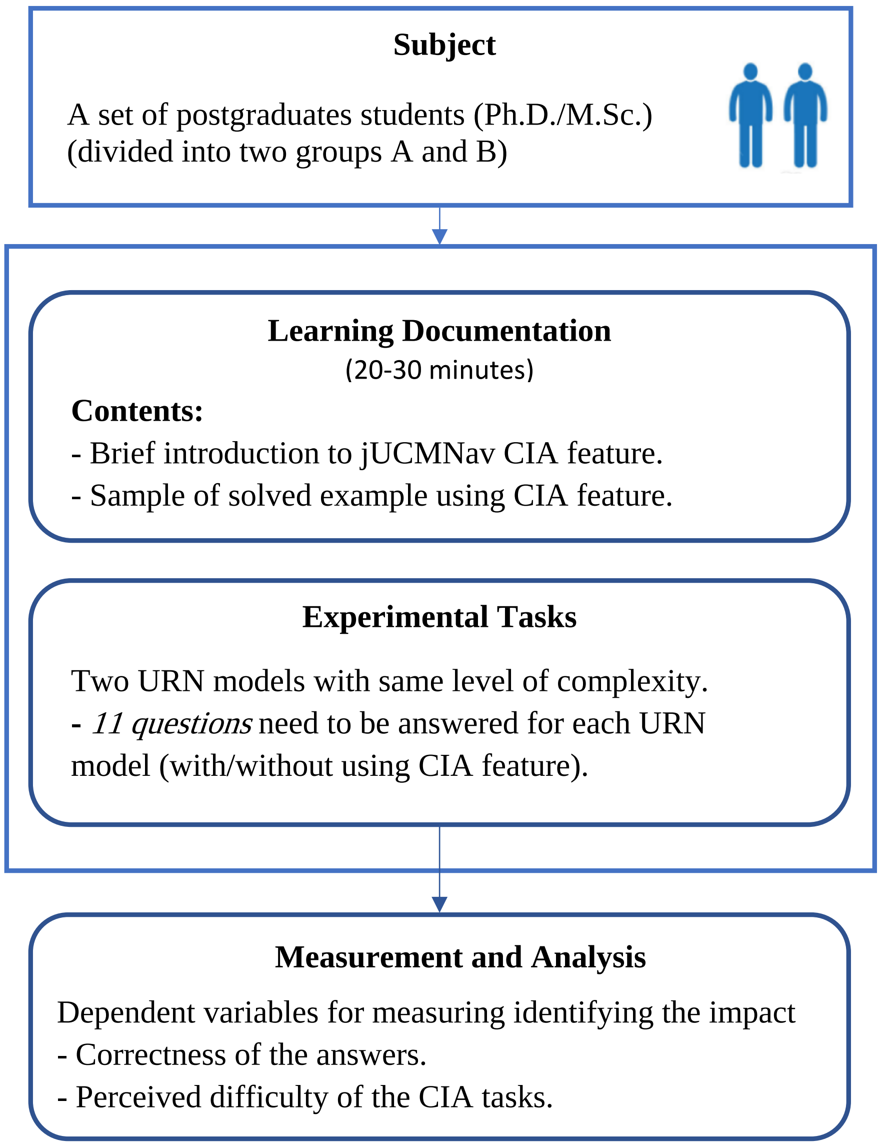

Subject
A set of postgraduates students (Ph.D./M.Sc.)
(divided into two groups A and B)
Learning Documentation
(20-30 minutes)
Contents:
- Brief introduction to jUCMNav CIA feature.
- Sample of solved example using CIA feature.
Experimental Tasks
Two URN models with same level of complexity.
- 11 questions need to be answered for each URN model (with/without using CIA feature).
Measurement and Analysis
Dependent variables for measuring identifying the impact
- Correctness of the answers.
- Perceived difficulty of the CIA tasks.

Figure 5.12: An overview of experimental plan

5.2.2 Context

The context of the experiment is two URN models namely, Adverse Event Management System and Commuting System. The models were evaluated by 10 subjects in order to identify the impacted of change without/with use of CIA feature.

5.2.3 Subjects

To provide more confidence on the results obtained by our approach, 10 members majoring in software engineering and computer science who have an experience with modeling, especially in URN language and familiarity with the jUCMNav tool. Also, they are unfamiliar with an automated analysis of CIA feature to URN model to provide their judgments on the results manually and by using our implemented feature. The members were divided into two groups randomly, each group was given five materials and those were distributed randomly for members.

5.2.4 Materials

The materials were divided into two parts, which are learning documentation and experimental tasks.

Learning documentation: This section provides two parts of URN model (i.e. GRL model and UCM model) to respondents with needed information for each part in order to carry out the experiment. We give about 20-30 minutes to read and understand the learning materials to perform the tasks. It consists of the following:

- An introduction to change impact analysis.
- An introduction to jUCMNav change impact analysis.
- Instructions that should be followed by subjects to carry out the experiment tasks.
- A generic example of the comprehension using change impact analysis feature.

Experimental Tasks: We summarized the given models to subjects in table 5.2.4. Each model has 11 questions of very similar complexity. Those questions are available online.

Group A
Case study 1: AEMS system. **eleven questions** need to be answered *without***using CIA feature**.
Case study 2: Commuting. **eleven questions** need to be answered **using CIA feature**.
Group B
Case study 1: Commuting. **eleven questions** need to be answered *without***using CIA feature**.
Case study 2: AEMS system.**eleven questions** need to be answered **using CIA feature**.

Table 5.8: Experiment Material

5.2.5 Variables

The dependent variables that are used to measure the identifying the impact of the URN model are (1) correctness of the answers, (2) perceived difficulty of the CIA task. The independent variable is performed the identifying tasks.

5.2.6 Hypotheses

Our experiment consists of two main hypotheses, stated in table 5.9. For each hypothesis, there are the null hypothesis, alternative hypothesis, and dependent variable. The first hypothesis is test whether there is a difference in the correctness of the answers when performed the identifying task manually or with CIA feature. The second hypothesis is test whether there is a difference in the perceived difficulty of the CIA tasks when performed manually or using the tool proposed.

Hypotheses
Hypothesis 1
Null hypothesis–$H_0 - 1$: There is no difference in the correctness of the answers when performed the identifying task manually or with CIA feature.
Alternative hypothesis–$H_1 - 1$: There is a difference in the correctness of the answers when performed the identifying task manually or with CIA feature.
Dependent variable: Correctness of the answers
Hypothesis 2
Null hypothesis–$H_0 - 2$: There is no difference in the perceived difficulty of the CIA tasks when performed manually or using the tool proposed.
Alternative hypothesis–$H_1 - 2$: There is a difference in the perceived difficulty of the CIA tasks when performed manually or using the tool proposed.
Dependent variable: Perceived difficulty of the CIA task.

Table 5.9: Set of hypotheses

5.2.7 Data analysis and interpretation

After completing the experiment tasks, we extracted the data that collected from subjects. We use the SPSS software [63] to illustrates statistical descriptive and perform t-test analysis of our hypotheses. For the first dependent variable (i.e. *Correctness*), we stated "1" for the correct answers, and "0" for the incorrect answers. Since we conduct tasks once with CIA feature and once with manual execution. We coded an automation execution as "1" and the manual execution as "0".

Table 5.10 provides the correctness cross tabulation analysis. In order to test the hypothesis $H_0 - 1$, we need to compute the cross-tabulation of the correctness of the answers on the use of CIA against the correctness of the obtained answers. The correct answers by using the CIA feature is (107) correct answers (97.3%) and the incorrect answers is (3) correct answers (2.7%), while the manual execution obtained (86) correct answers (78.2%) versus (24) incorrect answers (21.8%).

			Correctness		
			0	1	Total
CIAFeature	0	Count	24	86	110
		% within CIAFeature	21.8%	78.2%	100.0%
	1	Count	3	107	110
		% within CIAFeature	2.7%	97.3%	100.0%
Total		Count	27	193	220
		% within CIAFeature	12.3%	87.7%	100.0%

Table 5.10: CIA-Correctness Cross tabulation

Based on the provided result in Table 5.10, we can conclude that the use of CIA feature is strongly improved and increased the number of correct answers.

Furthermore, we apply independent t-test to the correctness as a test variable and the use of CIA feature as a group variable in order to prove that the improvement is significant. Levens test (see Table 5.11) shows the equality of variances is not assumed (Sig. $= 0.000 < \alpha = 0.05$). Based on the value of significance, we can conclude that there is a statistically significant difference between groups with respect to the correctness variable (with/without the use of CIA feature). Hence, we reject the null hypothesis $H_0 - 1$ and accept the alternative hypothesis $H_1 - 1$.

		Levene's Test for Equality of Variances		t-test for Equality of Means				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Correctness	Equal variances assumed	116.131	.000	-4.48940	218	.000	-.19091	.04252
	Equal variances not assumed			-4.48940	142.10338	.000	-.19091	.04252

Table 5.11: Test differences between means with respect to correctness (t-test)

The second hypothesis $H_0 - 2$, Table 5.12 shows the means of the perceived difficulty of the CIA task in both automation execution and manual execution. The mean of difficulty without the use of CIA feature is 2.727, where the mean with use of CIA feature is 1.3. According to the means values, we conclude that with use of CIA feature, the perceived difficulty of the CIA task decreased to the half.

	CIAFeature	N	Mean	Std. Deviation	Std. Error Mean
Difficulty	0	110	2.7273	1.03966	.09913
	1	110	1.3000	.47987	.04575

Table 5.12: Descriptive the perceived difficulty of the CIA task

Based on the value of Sig. for Levenes (see Table 5.13), the value of Sig. 0.000

which is less than 0.05, we can conclude that there is a difference in the perceived difficulty of the CIA tasks when performed manually or using the tool proposed.

		Levene's Test for Equality of Variances		t-test for Equality of Means				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Difficulty	Equal variances assumed	65.086	.000	13.073	218	.000	1.42727	.10918
	Equal variances not assumed			13.073	153.426	.000	1.42727	.10918

Table 5.13: Test differences between means with respect to the perceived difficulty of the CIA tasks (t-test)

5.2.8 Precision and Recall

For the validation, we asked experts to manually identify the impact of change and they were unware of CIA feature. The 3rd column and 4th column in tables 5.14 and 5.15 show the number of identified elements by the experts for AEMS model with respect to the slicing criterion and the number of identified elements by CIA feature, respectively. To provide more confidence on the results obtained by our proposed approach. We provided the experts with the URN model and asked them the following question: What is the impact of the change in URN model with respect to certain slicing criterion?

Further, the effectiveness of our approach is evaluated by recall and precision [62], two metrics that have been widely used in pattern recognition and information retrieval. Here, we borrow these two metrics from information retrieval, and adapt them to fit in with the evaluation model. We calculate the precision and recall of identifying the impacted elements on the model in order to assess the

Task	SC = (ElementName, ChangeType)	Identified by experts	Identified by CIA feature	Precision	Recall
1	(Fast Process, modify)	3	3	100%	100%
2	(High Data Quality, modify)	4	4	100%	100%
3	(Completeness index, delete)	8	8	100%	100%
4	(ContributionLink, delete)	3	3	100%	100%
5	(High Accuracy, modify)	6	6	100%	100%
				100%	100%

Table 5.14: AEMS GRL Model - Identified elements impacted w.r.t SC

Task	SC = (ElementName, ChangeType, SCVariables)	Identified by experts	Identified by CIA feature	Precision	Recall
1	(Number of events, add, -)	4	4	100%	100%
2	(WarnObserver, modify, EventReady)	2	2	100%	100%
3	(ScoreEvent, delete, -)	4	4	100%	100%
4	(DischargePatient, add, Discharged)	4	4	100%	100%
5	(AEMS-CreateVisit, delete, -)	17	17	100%	100%
				100%	100%

Table 5.15: AEMS UCM Model - Identified elements impacted w.r.t SC

accuracy of the tool, we need the following definitions:

- **True Positive (TP)**: A set of impacted elements are correctly identified.
- **False Positive (FP)**: A set of elements were identified as impacted while they are not.
- **False Negative (FN)**: A set of elements were not identified while they are impacted.

TP, FP, and FN are calculated at a coarse-grained level, meaning that the set of impacted elements are identified by experts and by CIA feature should exactly match, in terms of their methods and attributes. Precision assesses the number of the truly impacted elements among the number of all the impacted elements while recall assesses the number of identified impacted elements among the all impacted elements. Quantitative evaluation of the approache was conducted by precision and recall:

$$\boldsymbol{Precision} = \frac{\{TrueImpactedElements\} \cap \{AllImpactedElements\}}{\{TrueImpactedElements\}}$$

$$\boldsymbol{Recall} = \frac{\{TrueImpactedElements\} \cap \{AllImpactedElements\}}{\{AllImpactedElements\}}$$

Apparently, it can be shown from the tables 5.14 and 5.15 that the precision and recall of AEMS model are 100%. The impacted elements obtained by CIA feature are equal to the impacted elements provided by the experts.

CHAPTER 6

DISCUSSION

In what follows, we discuss the benefits and limitations of the proposed approach, then we compare it with related work.

6.1 General Benefits of the URN-based CIA Approach

The presented URN-based change impact analysis approach presents the following advantages:

- It helps maintainers and analysts answer *"what if... ?"* questions, and assess the consequences of changes in GRL and UCM specifications. Indeed, our approach provides an insight into how changes propagate within a GRL model, across models (i.e., from GRL to GRL) through URN Links, within a UCM model, across (i.e. from UCM to UCM), and across URN model (i.e. from GRL to UCM and vice versa). In addition, it allows for the identification of the impacted GRL strategies, if any. This would allow for reasoning about different alternatives, when it comes to implement changes in GRL models.

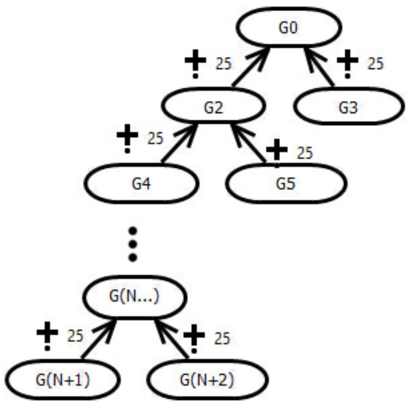
- Our approach handles some issues such as loop recognition and concurrency. For loop recognition, to avoid infinite loop within UCM model, we need to detect it during traversal. Since the order of execution scenario in a synchronous manner need to take into account, so we compute the all possible of execution and compute the dependencies for each execution.

- We have chosen GRL as target language, given its status as an international standard, but our proposed approach can likely be adapted and applied to other goal-oriented languages such as *i** [64] and TROPOS [65].

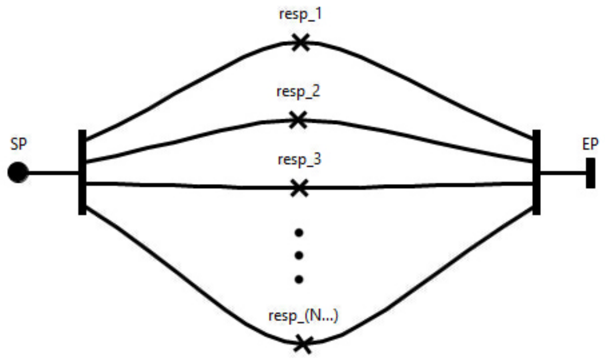
- Our approach is fully automated and covers the full GRL and UCM language constructs.

6.2 Salability

We also apply our CIA feature on two extreme cases for both models UCM and GRL (see Figures 6.1(a) and 6.1(b)) to demonstrate the scalability of the approach. GRL contains Intentional Elements (i.e. representing the number of elements N = 200) and GRL links (i.e. representing the number of connected links l = 199). The GRL model extreme case is created like a nearly complete tree, because we need to construct the GRL Model Dependency Graph (GMDG) (see Sect. 3.2) based on this model. Note that the GMDG graph will be constructed as a binary tree with the left depth from the left to the root. The depth of node is the number of edges from the node to the trees root node. In general, the worst-case number of the traversal is the depth from the selected node as SC to the root of GMDG tree which is $O(\log n)$. We have applied the CIA feature in different level of depth of GMDG graph when N = 10, N = 50, N = 100, N = 150 and N = 200. We have observed that there was no impact on the machine.



(a) GRL extreme case



(b) UCM extreme case

Figure 6.1: Extreme Cases of GRL and UCM

For UCM model, it contains a set of responsibilities (representing the number of responsibilities N = 150), as described previously in Sect. 4.3.5. In case we need to perform change impact analysis task on a huge number of responsibilities enclosed within AND-Fork and AND-Join, that is mean more CIA processes are required. It is worth noting that we select the parallelism to demonstrate the scalability of our approach because it is a challenge when computing the dependencies during the forward traversal and it needs more processes that lead to suffering from the overhead computation. In addition, the order of execution is important, it may have an impact on the rest of the model global data. So when computing the dependencies, our approach generates all possible sequences of the branches enclosed within concurrency to perform CIA for each sequence. We have applied the CIA feature with different responsibilities which was selected randomly. We have not observed any impact on the execution.

6.3 Limitations

The proposed CIA approach is subject to the following limitations:

- Our approach supports the evaluation of the impact of a single change at a time. Assessing the impact of simultaneous changes is left for future work.
- We perform a single iteration to follow the involved URL links. The potentially impacted GRL elements are not used as a source/target to explore more URN connections, if any. However, we believe that implementing a transitive chain should take into account the semantics of the URN Links (i.e., there should be a strong dependency that justifies the capture of the full ripple effect). This is out of the scope of this research.

- The applicability of our approach was demonstrated using three case studies and a mock system only. Bigger case studies should provide a better assessment of the effectiveness of our proposed approach.

6.4 Comparison with related work

Less work has been done on creating change impact techniques for GRL-based requirements, which is due to their abstract nature. Several source code based CIA techniques have been proposed in order to help software understanding, debugging, or repression test. Based on program source code, [3] conducted a survey of change impact analysis techniques. Later, the change impact analysis research has extended to other artifacts such as design, requirements, and testing. A taxonomy for software change impact analysis was developed by Lehnert [26] and a comprehensive literature review [27] of 150 studies was conducted that related to change impact analysis of source code, architecture [28, 29], miscellaneous artifacts (e.g., configuration files, bug trackers, documentation) [30], and requirements models [33]. In what follows, we survey and compare existing model-based change impact approaches with respect to the following criteria: Note that the criteria were extracted from A Taxonomy for Software Change Impact Analysis [26] and A review of software change impact analysis [27]

- ***Scope***: refers to the model used.

- ***Change***: refers to a type of changes such as *A-Addition*, *D-Deletion*, *M-Modification*(i.e. replacement, rename, change value, etc.)), or *UC-Unstructured* Change(i.e. CVS change records or Log file entries).

- ***Technique***: refers to what technique the impact analysis approach proposed or used to perform change analysis such as *TR-Traceability*, *DG-Dependency Graph*, *S-Model Slicing*, *DA- Dependecy Analysis*, *ER-Explicit Rule*, *IR- Information retrieval*, *ET-Execute trace*?

- ***Direction***: refers to the direction of traversal that used for the traceability.

- ***Dependency***: According to [66], many types of dependencies were proposed to compute the dependencies either D-Data flow or C-Control flow dependencies, etc. in our work well care only about supported dependencies.

- ***Purpose***: refer to the purpose of proposed work.

- ***Output***: refer to *HOW* the proposed work is presented.

- ***Output***: Availability of a tool.

Our proposed approach is listed in the last row in the table, as shown in Table 6.1. Change impact analysis [67] techniques have focused mainly on source code level [3] in order to help developers understand and maintain their programs. Less work has been devoted to change impact analysis in other software artifacts such as requirements and design models [26]. In what follows, we survey and compare existing goal-oriented CIA techniques with our proposed approach.

In a closely related work, Hassine [41] proposed a preliminary (and manual) CIA approach based on slicing GRL Model Dependency Graphs (GMDG). In this paper, we extend the approach by considering inter model propagation, GRL evaluation strategies, and URN Links. We have also fully automated it. Cleland-Huang et al. [38] introduced a probabilistic approach for managing the impact of a change using a Softgoal Interdependency Graph (SIG) that describes non-functional requirements and their dependencies. This technique allows for the analysis of the impact of changes by retrieving links between classes affected by changes in the SIG graph. Our approach is based on the GRL graph structure and does not distinguish between functional and non-functional requirements.

Tanabe et al. [68] introduced a change management technique in AGORA. The technique aims at detecting conflicts when a new goal is added and checks the satisfaction of the parent goal, when a goal is deleted. Semantic information, described as goal characteristics such as security or usability, should be attached to goals to allow for the detection of conflicts. Our approach considers structural change (both addition and deletion) propagation within the same model and across many models, regardless the semantic aspect of the impacted goals. Lee et al. [33] proposed a goal-driven traceability technique for analyzing requirements, which connects goals and use cases through three different traceability relations (evolution, dependency, and satisfaction), which are stored as a matrix. Impacted entities can then be identified by applying a reachability analysis on the matrix. Our GRL-based approach builds a GRL model dependency graph

(GMDG) to represent explicit and implicit, e.g., contribution, dependencies between model elements. In addition, our approach identifies the potential changes in other model elements that are linked through user-defined URN Links. Hassine et al. [31] proposed a change impact analysis approach for use case map that describes a scenarios dependencies. The dependencies between scenarios are classified as functional, containment and temporal dependencies which are used to identify the impact of change. However, the dependencies between scenarios are not part of the UCM model and the approach does not cover all UCM constructs. In this thesis, we extend the approach by considering inter model propagation, cover all UCM constructs, UCM data flow, and URN links between models. Alkaf et al. [14] introduced an automated GRL-based approach to change impact analysis. It helps the analyst to understand how a change is propagated within GRL model and a cross-related GRL model (i.e. from GRL to GRL), links using URN links. In this thesis, we extend their work, to assess the impact of such changes on related Use Case Maps (UCM) functional model.

Ernst et al. [37] proposed an approach to find suitable solutions (that minimize the effort required to implement new solutions) as requirements change. Their approach [37] explores a Requirements Engineering Knowledge Base (REKB), describing goals, tasks, refinements, and conflicts, in order to find new operations that are additionally required as a result of an unanticipated modification such as the addition of a new feature or the introduction of a new law. Our approach does simply spot potential impacted elements based on the GRL model structure and does not propose a solution to implement the change. In order to help developers identify where changes are required, Nakagawa et al. [39] proposed an approach based on the extraction of control loops, described as independent components that prevent the impact of a change from spreading outside them.

More recently, Grubb and Chechik [69] proposed an i*-based method to model the evolution of goal evaluations over time. Their proposed method integrates variability in intentions satisfaction (using qualitative values) over time allowing the stakeholders to understand and consider alternatives over time. In a closely related work to [69], Aprajita and Mussbacher [70] introduced TimedGRL, an extension of the GRL standard, allowing for the capture and analysis of a set of changes to a goal model over time (using quantitative values such as concrete dates). Both the goal model and the expected changes are represented in one model. However, both approaches described in [69] and [70] focus only on the evolution of satisfactions values (qualitative and quantitative) and they do not consider the evolution of the goal model structure over time.

In [71], the authors have proposed a new Activity-based Process Integration approach by giving a comprehensive evaluation of each new activity that might be added. But it still needed to improve by focusing on the real impact of changes for evaluation, automate the approach, and consider all type of changes. Our proposed approach allows for the identification of effects overall GRL models not only potential model.

Approach	Scope	Change	Technique	Dir.	Dep.	Purpose	Output	Tool
Hassine et al. [31]	UCM	A/D/M	S, DA	F	C	Comprehension	Reduced slice	-
Hassine [72]	GRL	A/D/M	TR, DG, S	B	C	Comprehension	Dependency Graph	-
Alkaf et al. [14]	GRL	A/D/M	ER, TR, DG, S	B	C, D	Comprehension	Closure slice	CAI GRL
Von Knethen A. [73]	Architecture	UC	RE, DA	-	D	Comprehension	Text Plain	-
Bai X. et al. [74]	UML	UC	DA	-	C, D	Testing	GUI tree structure in Java	Tool support
Settimi et al. [75]	UML	UC	TR, IR	-	-	Comprehension	Text Plain	-
Ecklund E. et al. [76]	Architecture	A	Traceability	-	C	Expected future changes	Text Plain	-
Briand et al. [7]	UML	UC	IR	-	-	Testing	UML Class diagram	-
Tanabe et al. [77]	Goal Model	A/D	DG	-	C	Testing (Detects conflicts)	UML Goal model	AGORA
Baslyman et al. [71]	URN	A	ET	F/B	C	enhancing existing system	URN model	jUCMNav
Cleland et al. [38]	Goal Model	A/D/M	TR, DA	-	C	detecting the impact change of NFR's	Goal model	-
Nakagawa et al. [39]	Goal Model	A	ET, TR	-	C	identify where change are required	graphical diagrams	k-tool
Briand et al. [78]	UML	A/D/M	IR, TR	-	C	Testing	UML Class diagram	iACMTool
Lionel B. et al. [79]	UML	Atomic change	TR, IR	-	C	Model Checking	UML Class diagram	VIATool
Lee et al. [33]	Goal Model	UC	TR, IR	-	C	Comprehension	-	-
Ernst et al. [37]	Goal Model	A	IR	-	-	enhancing existing system	-	-
Grubb et al. [80]	GRL	UC	TR, DA	-	-	Testing	-	-
Aprajita et al. [81]	GRL	UC	RT, DA	-	-	Model Checking	-	-
Arda et al. [82]	Architecture	UC	IR, DA	-	-	Model Checking	Text Plain	TRIC
Our work	URN	A/D/M	ER, S, TR, DG	F,B	C,D	Comprehension	Closure slice	CIAURN

Table 6.1: Comparison with related works

6.5 Threats to Validity

Like any empirical evaluation, our approach, the empirical validation and the performed experiment (empirical evaluation in Sect. 5.2) are subject to several limitations and threats to validity, categorized here according to three important types of threats identified by Wright et al. [83].

- *Construct validity*: a possible threat is that the empirical validation was performed with subjects who have a different level of knowledge; the time spent to complete the task depends on their experience with URN models and the time accuracy was not observed when reported. To avoid such threat, we selected subjects who have the same level of knowledge and experience in URN model, provided them with the same training materials in URN model and change impact analysis approach, and distributed them randomly into two groups.

- *Internal validity*: there is a risk threat to the empirical validation is that some participants did not answer seriously, or not finishing the task although they wrote down the start and end time of the task. This could be observed in future while performing the tasks. There is a possible risk of bias in the selection of URN model. To be not biased, we will use an existing URN models that are used for different projects. The size constraints and the availability of data expressions in the model were the criteria of selection models.

- *External validity*: the approach is currently tailored to URN. Although URN has many constructs that are common with other goal modeling languages or UML language, also, URN has some unique features (e.g., GRL evaluation strategies, UCM dynamic stubs, UCM executable scenarios, and a global data model). For that reason, the approach and guidelines might not be generalizable to other such languages without substantial adaptation. In addition, the small size of models and numbers of subjects might be a possible threat for the empirical evaluation and empirical validation. We can increase the level of confidence in the result by using models with large size and conduct an experiment with many participants.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we have presented an automated URN-based approach to change impact analysis. The proposed CIA approach allows maintainers and analysts understand how a change is propagated within a GRL model and across related GRL models (i.e., from GRL to GRL), linked using URN Links. In addition, the approach allows for the identification of the potentially impacted GRL evaluation strategies. Also, it helps maintainers and analysts understand how a change is propagated within UCM model and across related UCM models (i.e. from UCM to UCM) through URN links. Therefore, the approach combines CIA approach for both GRL and UCM by using URN links (i.e. GRL to UCM and UCM to GRL), such combination will contribute to the precision and completeness of requirements. The approach has been implemented as a feature within the jUCMNav [13] tool. Our proposed approach has been tested and evaluated using three public models and one mock model. In addition, we have conducted an experiment evolving 10 participants, the results show that there is a significant improvement of identifying impacted elements by using jUCMNav's new CIA feature. As a future work, we plan to extend our approach to cover simultaneous GRL / UCM changes and perform an iteration to follow the involved URL links. In addition to the proposed approach, we plan to investigate the implementation of the other techniques, such as dynamic forward change impact analysis with respect to the input. The potentially impacted URN elements are not used as a source/target to explore more URN connections, if any. The applicability of our approach was demonstrated using three case studies and a mock model only. Bigger case studies should provide a better assessment of the effectiveness of our proposed approach.

REFERENCES

[1] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman, "A requirements traceability to support change impact analysis," *Asian Journal of Information Tech*, vol. 4, no. 4, pp. 345–355, 2005.

[2] R. S. Arnold, *Software change impact analysis.* IEEE Computer Society Press, 1996.

[3] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013.

[4] Y. Li, J. Li, Y. Yang, and M. Li, "Requirement-centric traceability for change impact analysis: A case study," in *Making Globally Distributed Software Development a Success Story, International Conference on Software Process, ICSP 2008, Leipzig, Germany, May 10-11, 2008, Proceedings*, 2008, pp. 100–111. [Online]. Available: https://doi.org/10.1007/978-3-540-79588-9_10

[5] H. Zhang, J. Li, L. Zhu, R. Jeffery, Y. Liu, Q. Wang, and M. Li, "Investigating dependencies in software requirements for change

propagation analysis," *Information and Software Technology*, vol. 56, no. 1, pp. 40 – 53, 2014, special sections on International Conference on Global Software Engineering August 2011 and Evaluation and Assessment in Software Engineering April 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491300147X

[6] A. Goknil, I. Kurtev, K. van den Berg, and W. Spijkerman, "Change impact analysis for requirements: A metamodeling approach," *Information and Software Technology*, vol. 56, no. 8, pp. 950 – 972, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914000615

[7] L. C. Briand, Y. Labiche, and L. O'sullivan, "Impact analysis and change management of uml models," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on.* IEEE, 2003, pp. 256–265.

[8] A. Goknil, I. Kurtev, and K. van den Berg, "A rule-based change impact analysis approach in software architecture for requirements changes," *CoRR*, vol. abs/1608.02757, 2016. [Online]. Available: http://arxiv.org/abs/1608.02757

[9] A. von Knethen, "Change-oriented requirements traceability. support for evolution of embedded systems," in *International Conference on Software Maintenance, 2002. Proceedings.*, 2002, pp. 482–485.

[10] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman, "A software traceability validation for change impact analysis of object oriented software," in *Proceed-*

ings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006, Las Vegas, Nevada, USA, June 26-29, 2006, Volume 1, 2006, pp. 453–459.

[11] 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

[12] D. Amyot and G. Mussbacher, "User requirements notation: the first ten years, the next ten years," *JSW*, vol. 6, no. 5, pp. 747–768, 2011.

[13] "jucmnav project, v6.0.0," http://jucmnav.softwareengineering.ca, accessed: 2017-01-15.

[14] H. S. Alkaf, J. Hassine, A. Hamou-Lhadj, and L. Alawneh, *An Automated Change Impact Analysis Approach to GRL Models.* Cham: Springer International Publishing, 2017, pp. 157–172. [Online]. Available: https://doi.org/10.1007/978-3-319-68015-6_10

[15] L. Badri, M. Badri, and D. St-Yves, "Supporting predictive change impact analysis: a control call graph based technique," in *Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific.* IEEE, 2005, pp. 9–pp.

[16] B. Breech, M. Tegtmeyer, and L. Pollock, "Integrating influence mechanisms into impact analysis for increased precision," in *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on.* IEEE, 2006, pp. 55–65.

[17] L. Huang and Y.-T. Song, "Precise dynamic impact analysis with dependency analysis for object-oriented programs," in *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on.* IEEE, 2007, pp. 374–384.

[18] A. Orso, T. Apiwattanapong, and M. J. Harrold, "Leveraging field data for impact analysis and regression testing," in *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 5. ACM, 2003, pp. 128–137.

[19] J. Law and G. Rothermel, "Whole program path-based dynamic impact analysis," in *Proceedings of the 25th International Conference on Software Engineering.* IEEE Computer Society, 2003, pp. 308–318.

[20] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in *Proceedings of the 27th international conference on Software engineering.* ACM, 2005, pp. 432–441.

[21] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[22] S. N. Ahsan and F. Wotawa, "Impact analysis of scrs using single and multi-label machine learning classification," in *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement.* ACM, 2010, p. 51.

[23] L. C. Briand, J. Wust, and H. Lounis, “Using coupling measurement for impact analysis in object-oriented systems,” in *Software Maintenance, 1999.(ICSM’99) Proceedings. IEEE International Conference on.* IEEE, 1999, pp. 475–482.

[24] A. Beszedes, T. Gergely, S. Farago, T. Gyimothy, and F. Fischer, “The dynamic function coupling metric and its use in software evolution,” in *Software Maintenance and Reengineering, 2007. CSMR’07. 11th European Conference on.* IEEE, 2007, pp. 103–112.

[25] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, “Using information retrieval based coupling measures for impact analysis,” *Empirical software engineering*, vol. 14, no. 1, pp. 5–32, 2009.

[26] 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.* ACM, 2011, pp. 41–50.

[27] ——, “A review of software change impact analysis,” *Ilmenau University of Technology, Tech. Rep*, 2011.

[28] A. Aryani, I. D. Peake, M. Hamilton, H. Schmidt, and M. Winikoff, “Change propagation analysis using domain information,” in *Software Engineering Conference, 2009. ASWEC’09. Australian.* IEEE, 2009, pp. 34–43.

[29] Z. Xing and E. Stroulia, “Umldiff: an algorithm for object-oriented design differencing,” in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering.* ACM, 2005, pp. 54–65.

[30] G. Antoniol, V. F. Rollo, and G. Venturi, “Detecting groups of co-changing files in cvs repositories,” in *Principles of Software Evolution, Eighth International Workshop on.* IEEE, 2005, pp. 23–32.

[31] J. Hassine, J. Rilling, J. Hewitt, and R. Dssouli, “Change impact analysis for requirement evolution using use case maps,” in *Eighth International Workshop on Principles of Software Evolution (IWPSE’05).* IEEE, 2005, pp. 81–90.

[32] J. Hewitt and J. Rilling, “A light-weight proactive software change impact analysis using use case maps,” in *Software Evolvability, 2005. IEEE International Workshop on.* IEEE, 2005, pp. 41–46.

[33] 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, pp. 878–908, 2010.

[34] J. S. O’Neal, “Analyzing the impact of changing requirements,” in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM’01).* IEEE Computer Society, 2001, p. 190.

[35] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. Leite, "From goals to high-variability software design," in *International Symposium on Methodologies for Intelligent Systems.* Springer, 2008, pp. 1–16.

[36] A. Van Lamsweerde, "From system goals to software architecture," in *Formal Methods for Software Architectures.* Springer, 2003, pp. 25–43.

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

[38] 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.* ACM, 2005, pp. 362–371.

[39] 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.* IEEE, 2013, pp. 155–164.

[40] A. van Lamsweerde, "Requirements engineering: from craft to discipline," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering.* ACM, 2008, pp. 238–249.

[41] J. Hassine, "Change impact analysis approach to grl models," in *SOFTENG 2015: The First International Conference on Advances and Trends in Software Engineering.* ACM, 2015, pp. 1–6.

[42] M. S. Kilpinen, "The emergence of change at the systems engineering and software design interface," Ph.D. dissertation, University of Cambridge, 2008.

[43] H. Agrawal and J. R. Horgan, "Dynamic program slicing," in *Acm Sigplan Notices*, vol. 25, no. 6. ACM, 1990, pp. 246–256.

[44] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 1, pp. 26–60, 1990.

[45] M. Kamkar, "An overview and comparative classification of program slicing techniques," *Journal of Systems and Software*, vol. 31, no. 3, pp. 197–214, 1995.

[46] B. Korel and J. Laski, "Dynamic slicing of computer programs," *Journal of Systems and Software*, vol. 13, no. 3, pp. 187–195, 1990.

[47] M. L. Lee *et al.*, *Change impact analysis of object-oriented software.* George Mason University Virginia, 1998.

[48] M. Lee, A. J. Offutt, and R. T. Alexander, "Algorithmic analysis of the impacts of changes to object-oriented software," in *Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. Proceedings. 34th International Conference on.* IEEE, 2000, pp. 61–70.

[49] B. G. Ryder and F. Tip, "Change impact analysis for object-oriented programs," in *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering.* ACM, 2001, pp. 46–53.

[50] L. Liu and E. Yu, "Designing information systems in social context: a goal and scenario modelling approach," *Information systems*, vol. 29, no. 2, pp. 187–203, 2004.

[51] M. Weiss and D. Amyot, "Business process modeling with urn," *International Journal of E-Business Research (IJEBR)*, vol. 1, no. 3, pp. 63–90, 2005.

[52] 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.*, 2014. [Online]. Available: http://ceur-ws.org/Vol-1157/paper22.pdf

[53] 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

[54] J.-F. Bergeretti and B. A. Carré, "Information-flow and data-flow analysis of while-programs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 1, pp. 37–61, 1985.

[55] J. Silva, "A vocabulary of program slicing-based techniques," *ACM computing surveys (CSUR)*, vol. 44, no. 3, p. 12, 2012.

[56] O. Akhigbe, D. Amyot, A. A. Anda, L. Lessard, and D. Xiao, "Consistency analysis for user requirements notation models." in *iStar*, 2016, pp. 43–48.

[57] A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, and A. J. Forster, “Business process management with the user requirements notation,” *Electronic Commerce Research*, vol. 9, no. 4, pp. 269–316, 2009.

[58] B. Kitchenham, S. L. Pfleeger, L. Pickard, P. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, “Preliminary guidelines for empirical research in software engineering,” *IEEE Trans. Software Eng.*, vol. 28, no. 8, pp. 721–734, 2002. [Online]. Available: http://dx.doi.org/10.1109/TSE.2002.1027796

[59] A. Jedlitschka and D. Pfahl, “Reporting guidelines for controlled experiments in software engineering,” in *2005 International Symposium on Empirical Software Engineering (ISESE 2005), 17-18 November 2005, Noosa Heads, Australia*, 2005, pp. 95–104. [Online]. Available: http://dx.doi.org/10.1109/ISESE.2005.1541818

[60] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[61] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[62] J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel *et al.*, “Performance measures for information extraction,” in *Proceedings of DARPA broadcast news workshop*, 1999, pp. 249–252.

[63] IBM, “SPSS software,” 2012. [Online]. Available: http://www-01.ibm.com/software/analytics/spss/

[64] E. S. Yu, “Towards modelling and reasoning support for early-phase requirements engineering,” in *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on.* IEEE, 1997, pp. 226–235.

[65] P. Giorgini, J. Mylopoulos, and R. Sebastiani, “Goal-oriented requirements analysis and reasoning in the tropos methodology,” *Eng. Appl. Artif. Intell.*, vol. 18, pp. 159–171, March 2005.

[66] V. Ganapathy and S. Ramesh, “Slicing synchronous reactive programs,” *Electronic Notes in Theoretical Computer Science*, vol. 65, no. 5, pp. 50–64, 2002.

[67] S. Bohner and R. Arnold, “Ieee computer society press,” *Los Alamitos, CA, USA*, 1996.

[68] D. Tanabe, K. Uno, K. Akemine, T. Yoshikawa, H. Kaiya, and M. Saeki, “Supporting requirements change management in goal oriented analysis,” in *16th IEEE International Requirements Engineering (RE’08).* IEEE, 2008, pp. 3–12.

[69] A. M. Grubb and M. Chechik, “Looking into the crystal ball: Requirements evolution over time,” in *24th IEEE International Requirements Engineering Conference (RE’16)*, Sept 2016, pp. 86–95.

[70] Aprajita and G. Mussbacher, "TimedGRL: Specifying goal models over time," in *24th IEEE International Requirements Engineering Conference Workshops (REW)*, Sept 2016, pp. 125–134.

[71] M. Baslyman, B. Almoaber, D. Amyot, and E. M. Bouattane, *Activity-based Process Integration in Healthcare with the User Requirements Notation.* Cham: Springer International Publishing, 2017, pp. 151–169. [Online]. Available: https://doi.org/10.1007/978-3-319-59041-7_9

[72] J. Hassine, "Change impact analysis approach to grl models," 2015.

[73] A. v. Knethen, "A trace model for system requirements changes on embedded systems," in *Proceedings of the 4th International Workshop on Principles of Software Evolution*, ser. IWPSE '01. New York, NY, USA: ACM, 2001, pp. 17–26. [Online]. Available: http://doi.acm.org/10.1145/602461.602465

[74] X. Bai, W.-T. Tsai, R. Paul, K. Feng, and L. Yu, "Scenario-based modeling and its applications," in *Object-Oriented Real-Time Dependable Systems, 2002.(WORDS 2002). Proceedings of the Seventh International Workshop on.* IEEE, 2002, pp. 253–260.

[75] R. Settimi, J. Cleland-Huang, O. B. Khadra, J. Mody, W. Lukasik, and C. DePalma, "Supporting software evolution through dynamically retrieving traces to uml artifacts," in *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of.* IEEE, 2004, pp. 49–54.

[76] E. F. Ecklund, Jr., L. M. L. Delcambre, and M. J. Freiling, “Change cases: Use cases that identify future requirements,” *SIGPLAN Not.*, vol. 31, no. 10, pp. 342–358, Oct. 1996. [Online]. Available: http://doi.acm.org/10.1145/236338.236372

[77] D. Tanabe, K. Uno, K. Akemine, T. Yoshikawa, H. Kaiya, and M. Saeki, “Supporting requirements change management in goal oriented analysis,” in *2008 16th IEEE International Requirements Engineering Conference*, Sept 2008, pp. 3–12.

[78] L. Briand, Y. Labiche, L. OSullivan, and M. Swka, “Automated impact analysis of uml models,” *Journal of Systems and Software*, vol. 79, no. 3, pp. 339 – 352, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412120500097X

[79] L. C. Briand, Y. Labiche, and T. Yue, “Automated traceability analysis for uml model refinements,” *Information and Software Technology*, vol. 51, no. 2, pp. 512 – 527, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584908000943

[80] A. M. Grubb and M. Chechik, “Looking into the crystal ball: Requirements evolution over time,” in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, Sept 2016, pp. 86–95.

[81] G. Mussbacher *et al.*, “Timedgrl: Specifying goal models over time,” in *Requirements Engineering Conference Workshops (REW), IEEE International.*

IEEE, 2016, pp. 125–134.

[82] A. Goknil, I. Kurtev, K. van den Berg, and W. Spijkerman, "Change impact analysis for requirements: A metamodeling approach," *Information and Software Technology*, vol. 56, no. 8, pp. 950 – 972, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914000615

[83] H. K. Wright, M. Kim, and D. E. Perry, "Validity concerns in software engineering research," in *Proceedings of the FSE/SDP workshop on Future of software engineering research.* ACM, 2010, pp. 411–414.

Vitae

- Name: Hasan Salem Omar Al-Kaf
- Nationality: Yemeni
- Date of Birth: 7/10/1987
- Email: *hsoalkaff@gmail.com*
- Permenant Address: Al Khobar, Eastern Region, Saudi Arabia

- Academic Background: Bachelor degree in Computer Information System, Al al-Bayte University Jordan , currently I'm pursuing my master degree in Software Engineering at King Fahd University of Petroleum and Minerals - Saudi Arabia.
- Published paper: An automated Change Impact Analysis to GRL Models. Sept. 7, 2017

Appendix A

This appendix shows the result of applying CIA with respect to our test cases listed in 5.1.1.

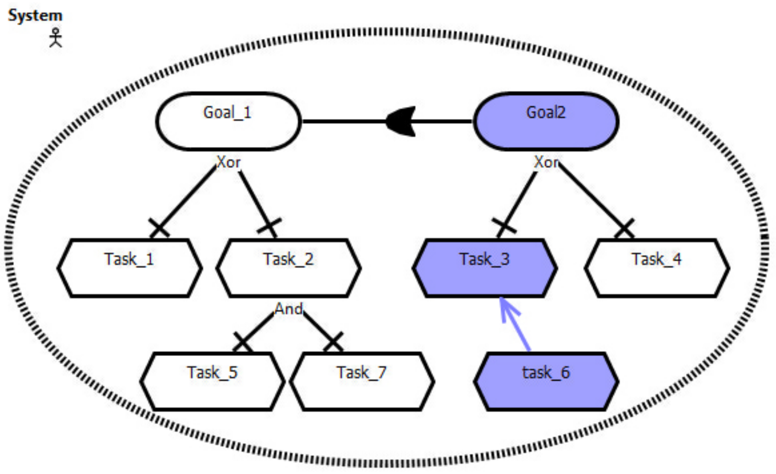


Figure 7.1: Impacted elements with respect toSC = ($task\_6$ in Graph (see fig. 5.1), *Addition*, -)

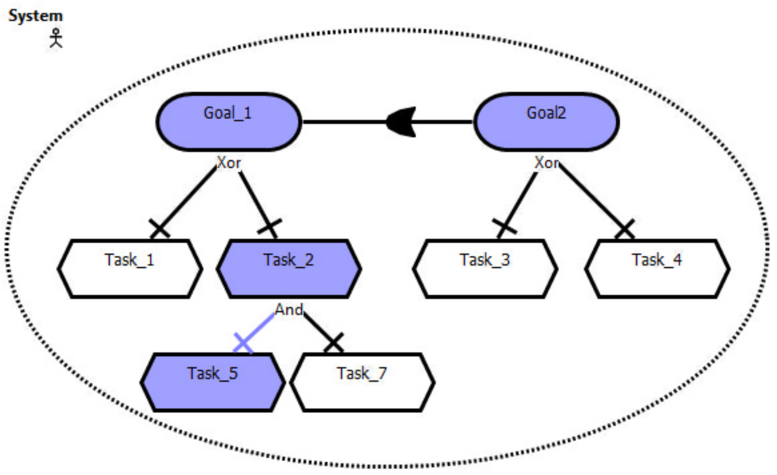


Figure 7.2: Impacted elements with respect toSC = ($task\_5$ in Graph (see fig. 5.1), *Deletion*, -)

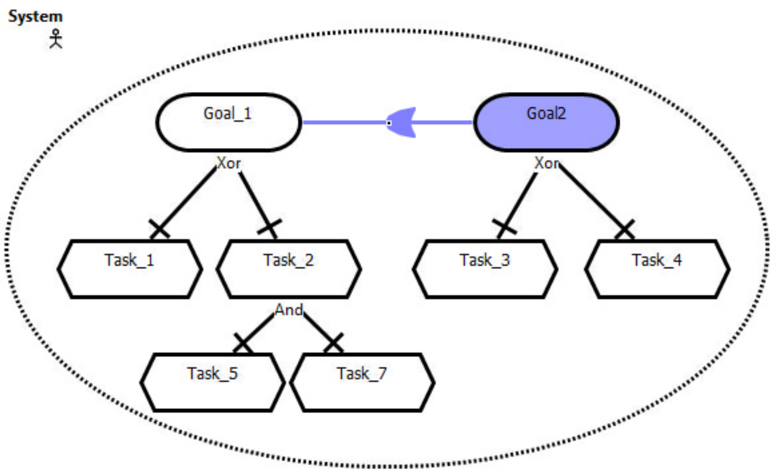
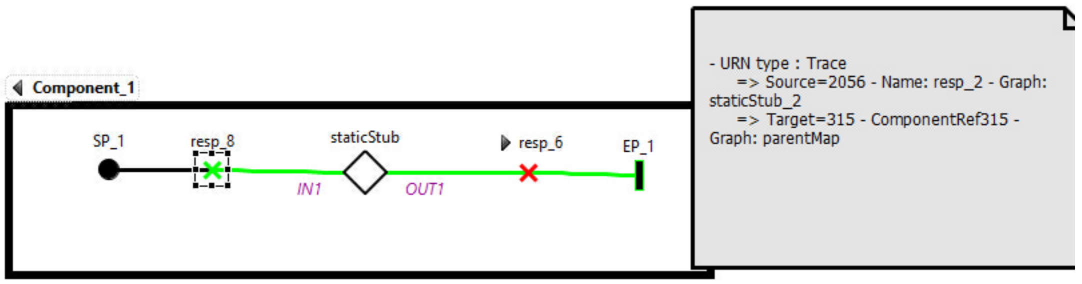
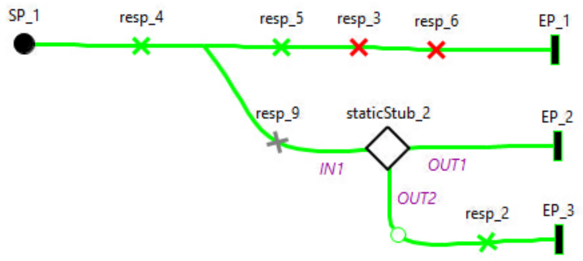


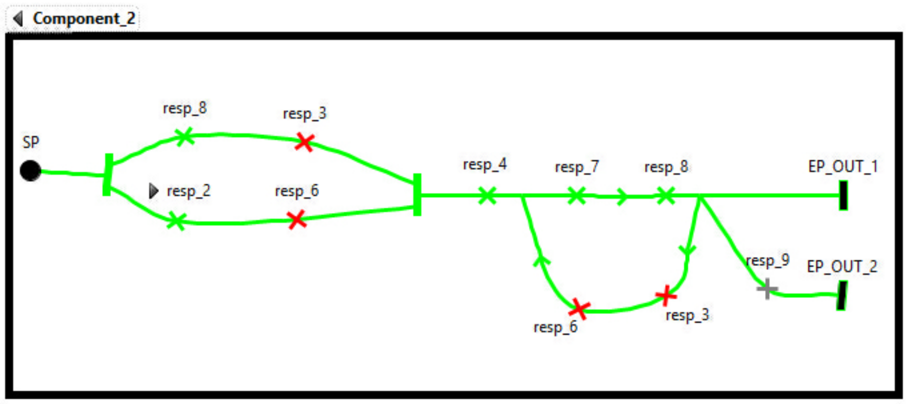
Figure 7.3: Impacted elements with respect toSC = ($DependencyLink$ in Graph (see fig. 5.1), *Modification*, -)


- URN type : Trace
    => Source=2056 - Name: resp_2 - Graph: staticStub_2
    => Target=315 - ComponentRef315 - Graph: parentMap

(a) Mian Map

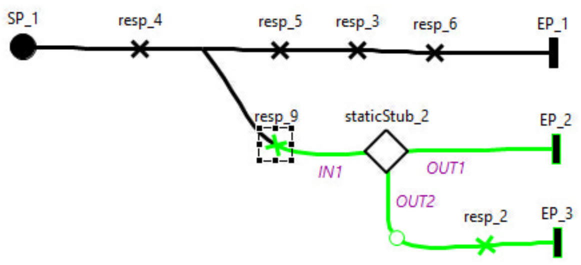


(b) staticStubMap - Plugin

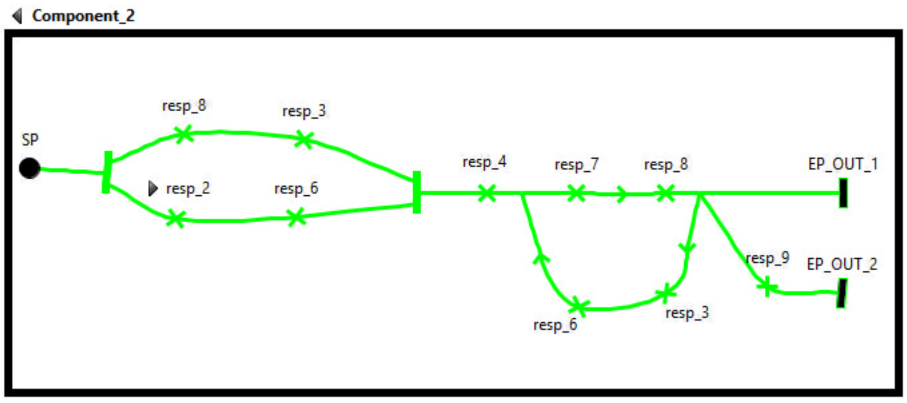


(c) staticStub_2 - Plugin

Figure 7.4: Impacted elements with respect toSC = (*resp_8* in MainMap (see fig. 5.2(a)), *Modification*, y)

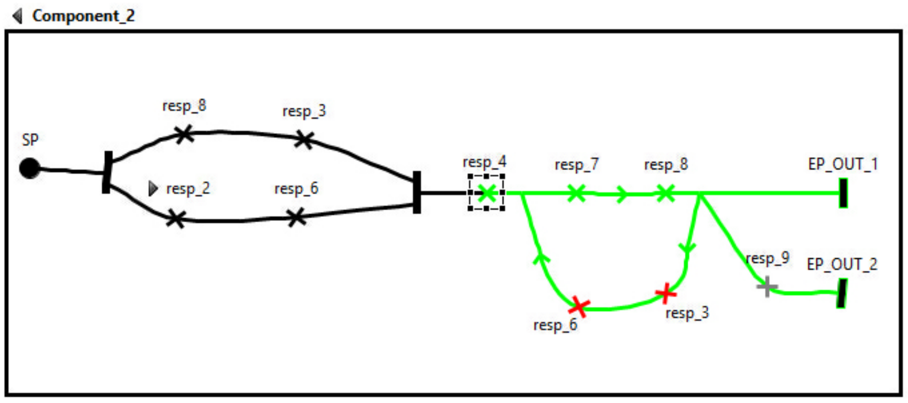


(a) staticStub Map



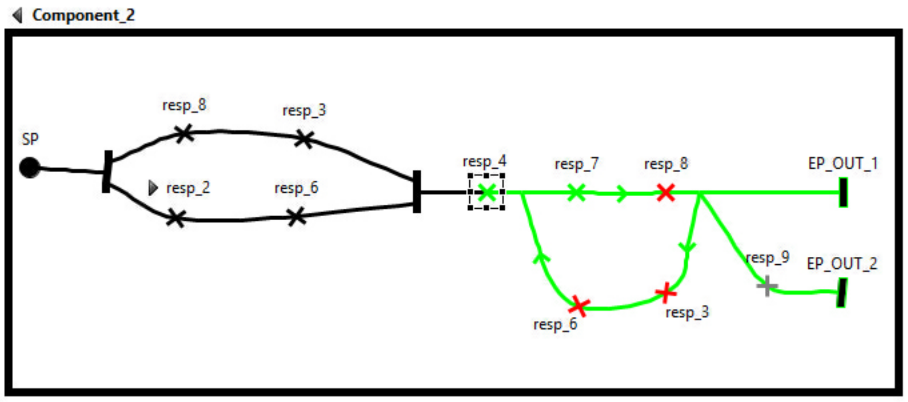
(b) staticStub_2 - Plugin

Figure 7.5: SC = (*resp_9* in staticStubMap (see fig. 5.2(b)), *Addition*, -)



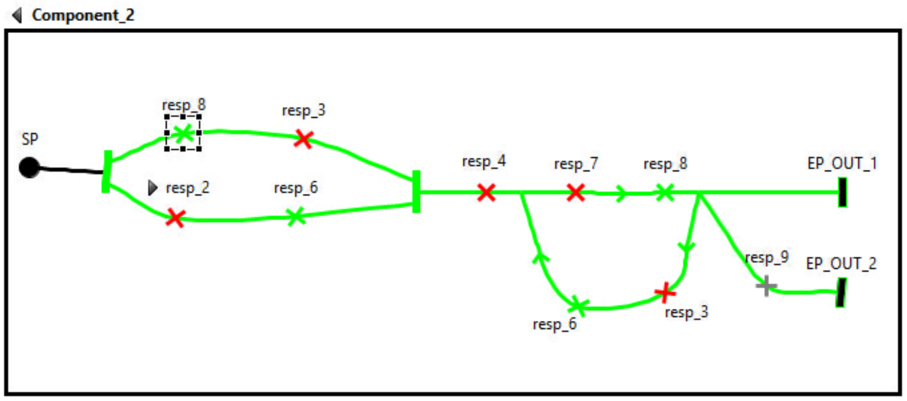
(a) staticStub_2 - Plugin

Figure 7.6: SC = (*resp_4* in staticStub_2 - Plugin (see fig. 5.2(c)), *Modification*, *(z,y)*)



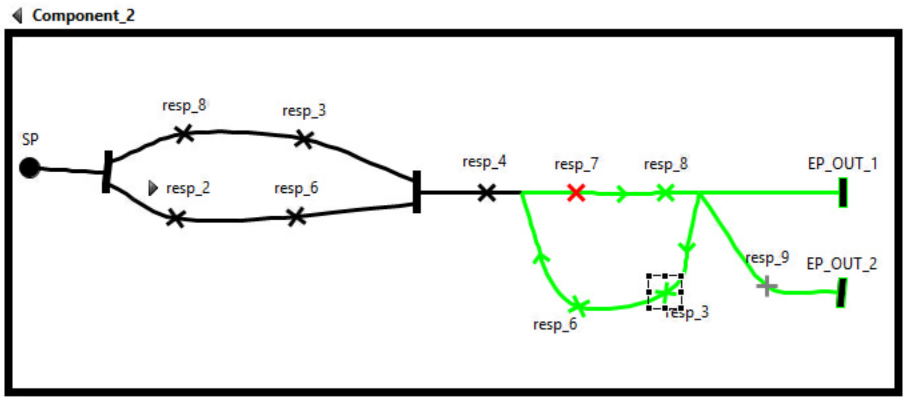
(a) staticStub_2 - Plugin

Figure 7.7: SC = (*resp_4* in staticStub_2 - Plugin (see fig. 5.2(c)), *Deletion*, z)



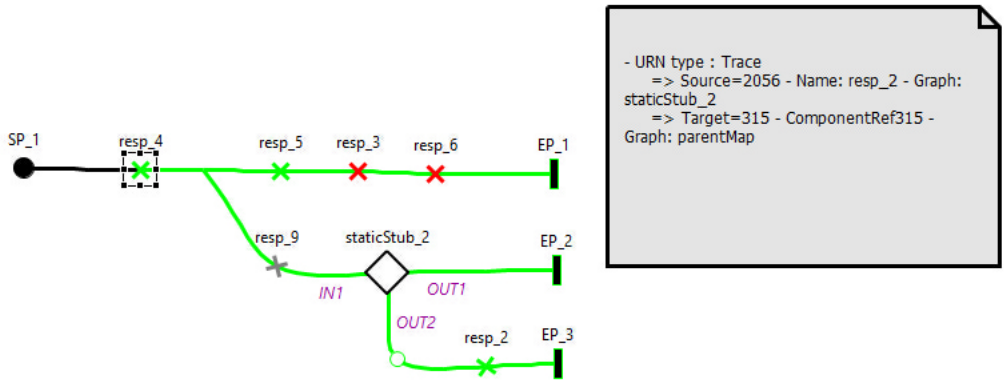
(a) staticStub_2 - Plugin

Figure 7.8: SC = (*resp_8* in staticStub_2 - Plugin (see fig. 5.2(c)), *Deletion*, *(t,n)*)



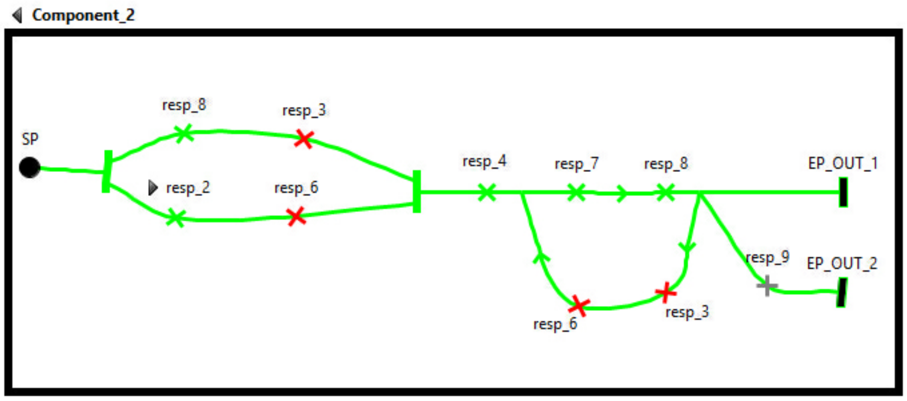
(a) staticStub_2 - Plugin

Figure 7.9: SC = (*resp_3* in staticStub_2 - Plugin (see fig. 5.2(c)), *Addition*, *t*)



- URN type : Trace
    => Source=2056 - Name: resp_2 - Graph: staticStub_2
    => Target=315 - ComponentRef315 - Graph: parentMap

(a) staticStub Map



(b) staticStub_2 - Plugin

Figure 7.10: SC = (*resp_4* in staticStubMap (see fig. 5.2(b)), *Modification*, *(z,y)*)

Appendix B

jUCMNav: A Change Impact Analysis (*CIA*) to User Requirements Notation (*URN*) Feature

Part 1: Learning Documentation of *URN* + Generic Example.

Part 1.1: A change impact analysis to *GRL* feature.

Consider the GRL example Fig. 1. The model has a URN link between ***Goal_4*** and ***Task_3***.

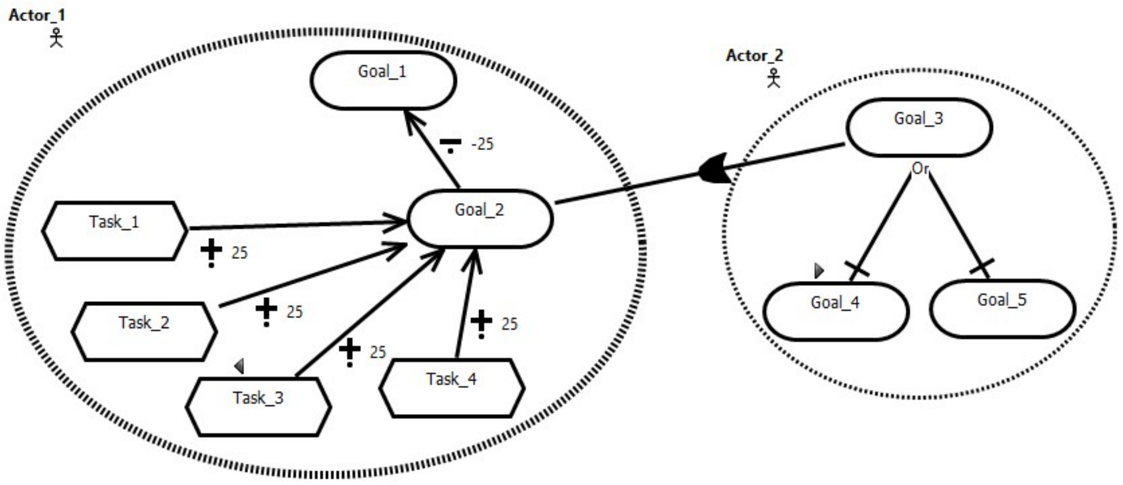


Figure 1 GRL Example

Three types of changes are implemented: ***Addition, Deletion*** and ***Modification***. These changes are applicable to ***Intentional Element*** (Goal, Softgoal, Task, and Indicator) or ***Links*** (Contribution, Correlation, and Dependency). Suppose that user is planning to make a change to an intentional element or a link and he wants to know all parts that might be affected by this change. To do so, the user may execute the GRL change impact analysis (CIA) feature by:

1. Choosing the intentional element or link of interest, for example **Task_3**, right click on it, then select what type of change you are planning (*Addition, Deletion* or *Modification*) from the sub-menu ***Change Impact Analysis***. In our example, we will select ***Addition***. As shown below in fig. 2.

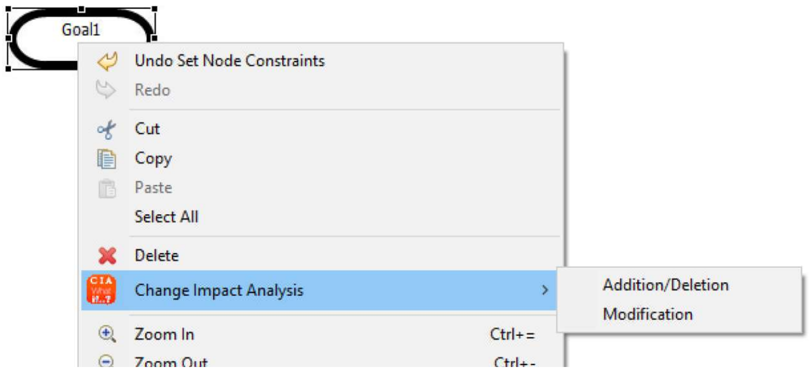
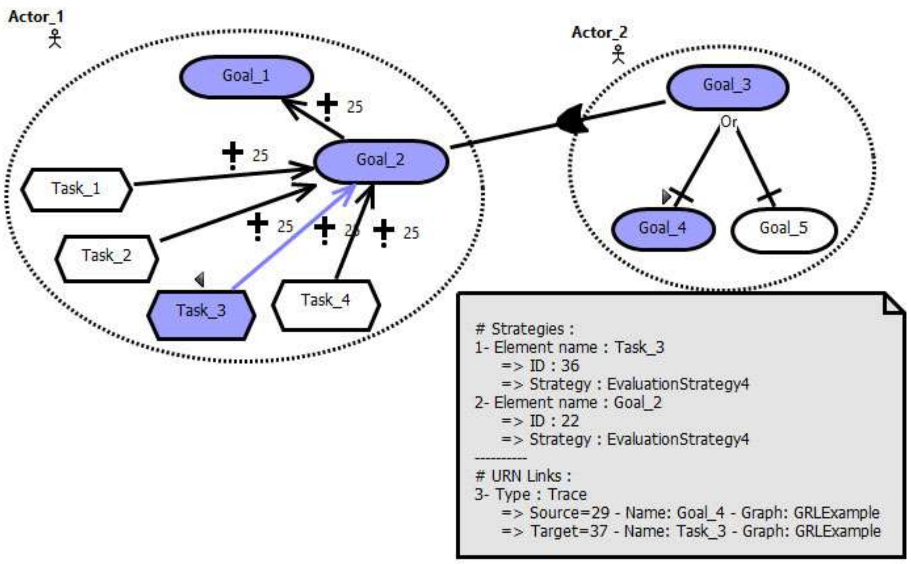


Figure 2 calling the GRL CIA Feature

2. Figure 3 shows all impacted elements in the GRL model by coloring them with ***purple*** color. In addition, it shows the URN Link details if exist between URN elements such as source/target of impacted elements and the name of graph which belongs to. In addition to URN Links, GRL CIA feature lists all the strategies that the impacted elements belong to.



```
# Strategies :
1- Element name : Task_3
    => ID : 36
    => Strategy : EvaluationStrategy4
2- Element name : Goal_2
    => ID : 22
    => Strategy : EvaluationStrategy4
----------
# URN Links :
3- Type : Trace
    => Source=29 - Name: Goal_4 - Graph: GRLExample
    => Target=37 - Name: Task_3 - Graph: GRLExample
```

Figure 3 All impacted elements, URN links, and strategies.

<u>Part 1.2</u>: A change impact analysis to *UCM* models.

Consider the UCM in Figure 4. The model contains a set of responsibilities. Note that each responsibility is named with its code to facilitate the reading of the code embedded within responsibilities. Responsibility R4 does not have code.

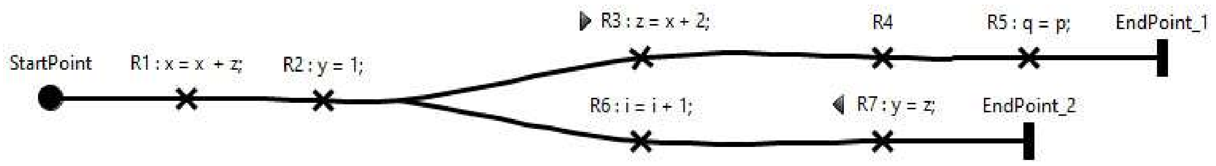


Figure 4 UCM Example

Three types of changes (***Addition, Deletion*** and ***Modification***) are supported on ***responsibilities*** and ***or-Fork branches, where the user can target specific variables, if any. It is worth noting that*** *addition* and *deletion* do not require variables' selection.

In order to compute the set of impacted elements, the user may execute the UCM change impact analysis feature by:

1. Choosing the ***Responsibility*** or ***Or-Fork branch, such as*** **R1** : x = x + z, right click on it, then select what type of change is planned (*Addition, Deletion* or *Modification*) from the submenu of ***Change Impact Analysis***. In this example, we will select ***Modification***. As shown in Fig. 5.

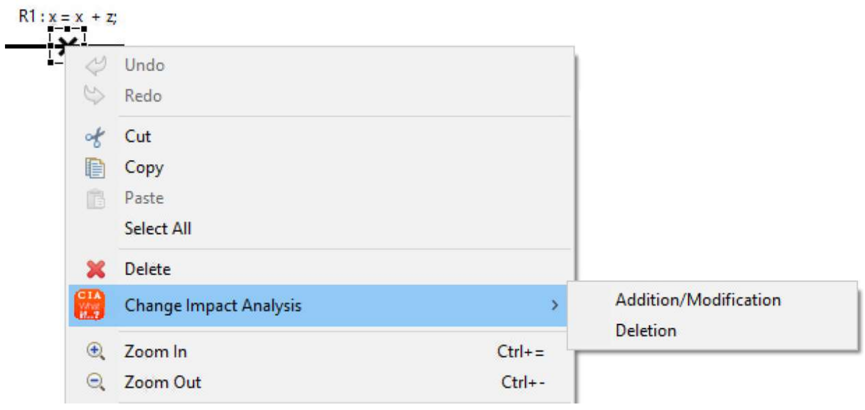


Figure 5 Calling UCM CIA Feature

2. After selecting the type of change, the selection criteria window will appear. In the **left** box, all variables reside in the chosen criterion are listed. Then, the user may select these variables from left box to move them to **right** box "Selected variables" by using the arrow button. In our example we will select all variables (i.e. x, z).

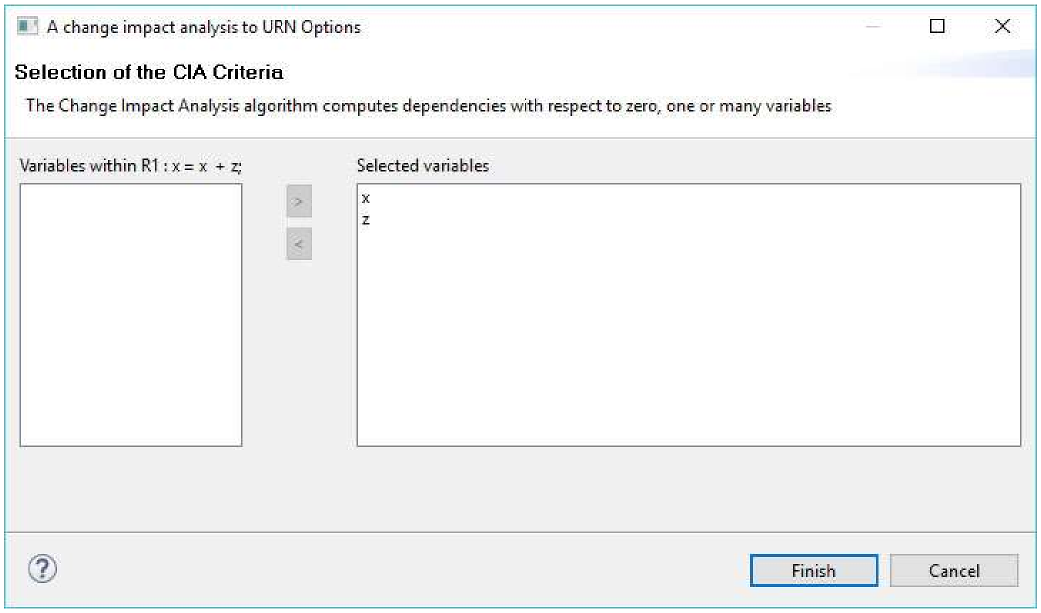
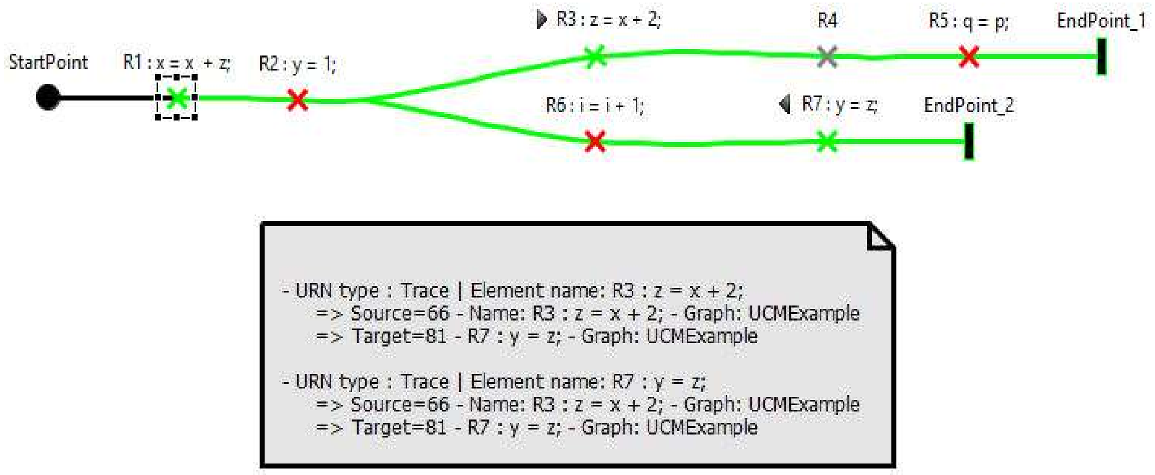

A change impact analysis to URN Options
Selection of the CIA Criteria
The Change Impact Analysis algorithm computes dependencies with respect to zero, one or many variables
Variables within R1 : x = x + z;
Selected variables
x
z
>
<
Finish
Cancel

Figure 6 Selection criteria window

3. Figure 7 shows the impacted elements (marked with a different color) as a result of the selected type of change. Note that the impacted elements are marked in green color, the elements that are not impacted are colored in red color, and the elements that do not have any embedded code are colored in gray color. All impacted URN links are also displayed in a URN comment box.



```
- URN type : Trace | Element name: R3 : z = x + 2;
    => Source=66 - Name: R3 : z = x + 2; - Graph: UCMExample
    => Target=81 - R7 : y = z; - Graph: UCMExample

- URN type : Trace | Element name: R7 : y = z;
    => Source=66 - Name: R3 : z = x + 2; - Graph: UCMExample
    => Target=81 - R7 : y = z; - Graph: UCMExample
```

Figure 7 Marked impacted Element

Part 2: Comprehension of URN model. (Group A)

Part 2.1: Comprehension of URN model without using the CIA feature.

Consider the following URN model of **an adverse event management system system** (AEMS). See Figures 8, 9, 10, and 11.

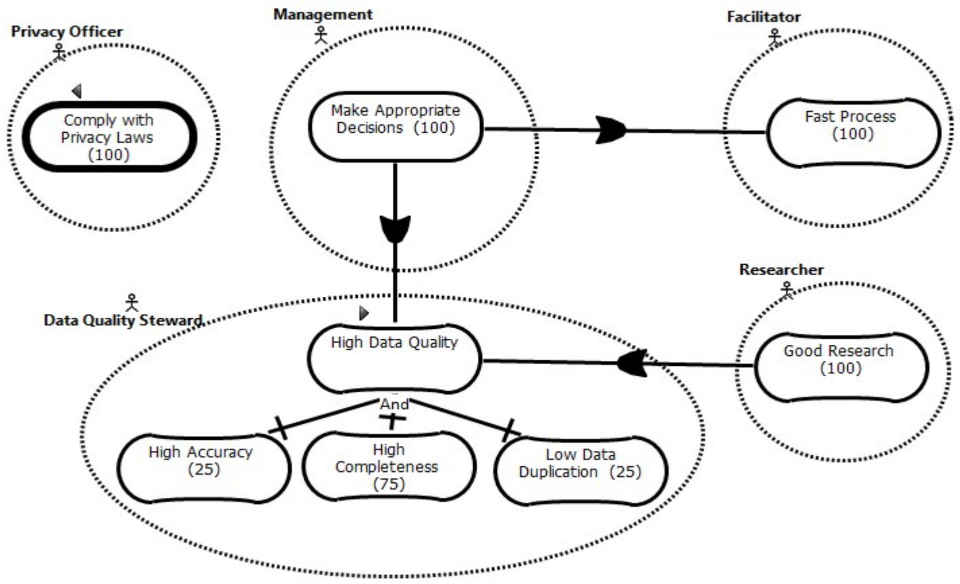


Figure 8 "Goals" Map – GRL

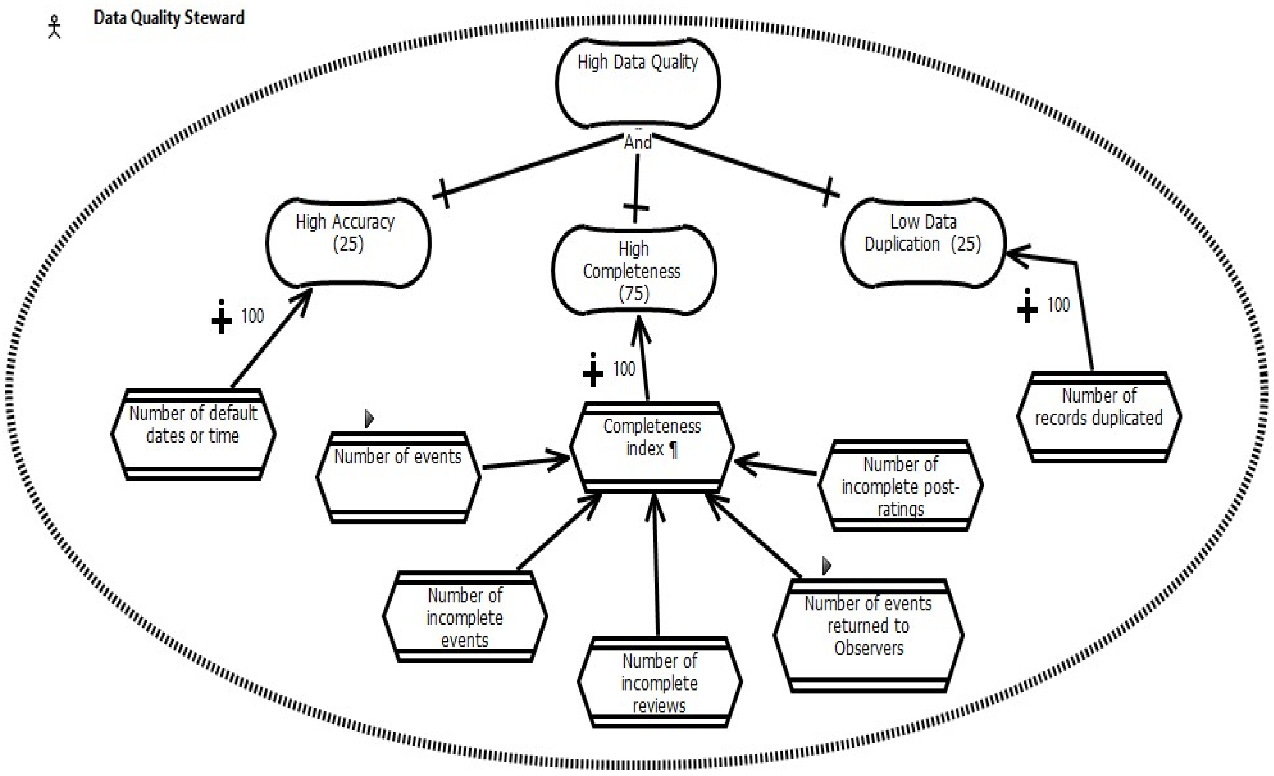

Data Quality Steward
High Data Quality
And
High Accuracy (25)
High Completeness (75)
Low Data Duplication (25)
100
100
100
Number of default dates or time
Number of events
Completeness index ¶
Number of incomplete post-ratings
Number of records duplicated
Number of incomplete events
Number of incomplete reviews
Number of events returned to Observers

Figure 9 "DQS-KPI" Map - GRL

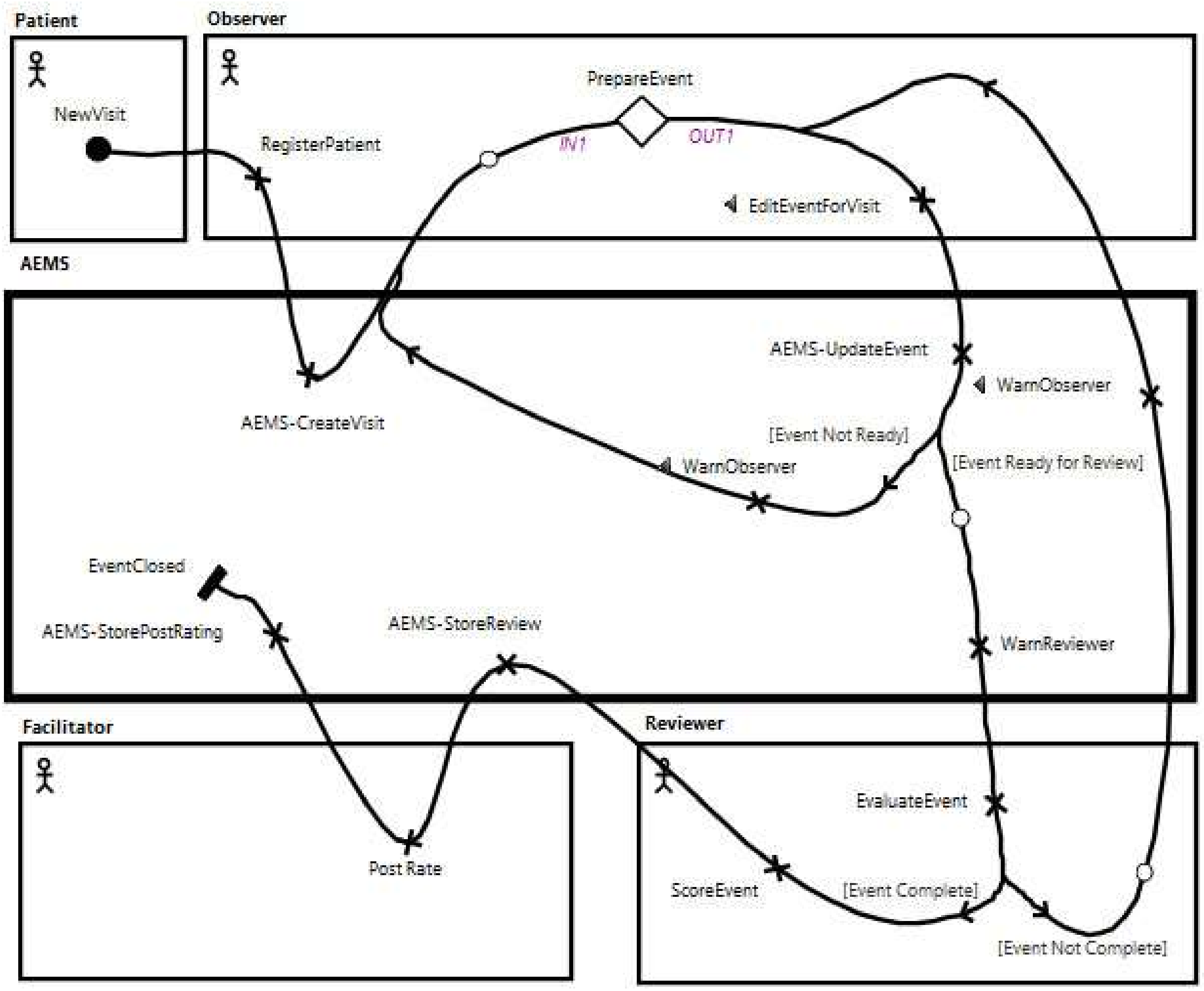


Figure 10 "Process" Map – UCM

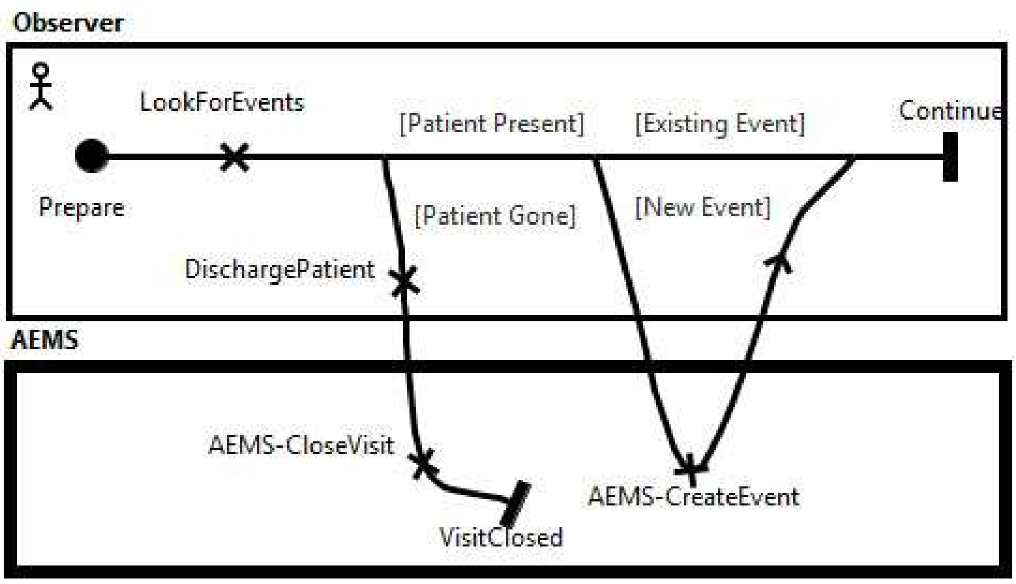


Figure 11 "Prepare Event" plug-in Map – UCM

Please answer the following questions and note the time spent in answering each question:

Q1: [**Fig. 8**] We plan to add a new ***Softgoal*** and a new positive contribution link to the softgoal ***High Completeness*** in **Goals map**.

Q1.1: Please mark, on the model, all elements and links that might be impacted by this modification. (Use a different color):

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]
Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q1.2: List any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element.

. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q2: [**Fig. 9**] We plan to delete the GRL Indicator ***Number of events returned to Observers*** in **DQS-KPI map**.
Q2.1: Please mark, on the model, all elements and links that might be impacted by this modification. (Use a different color)

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q2.2: List any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element.

. . . .
. . . .
. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………]   End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy   [ ] Easy   [ ] Neither easy nor difficult   [ ] Difficult   [ ] Very Difficult

Q3: [**Fig. 9**] We plan to modify ***the Contribution link*** between ***Number of records duplicated or time*** and ***Low Data Duplication***) in **DQS-KPI map**. Will the softgoal ***High Completeness*** be impacted?

Yes [ ]   No [ ]

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………]   End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy   [ ] Easy   [ ] Neither easy nor difficult   [ ] Difficult   [ ] Very Difficult

Q4: [**Fig. 10**] We plan to delete the responsibility ***RegisterPatient*** in **Process Map** (within Observer component).

Q4.1: Please draw a ***circle*** on all the elements that might be impacted by deleting this responsibility?

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………]   End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy   [ ] Easy   [ ] Neither easy nor difficult   [ ] Difficult   [ ] Very Difficult

Q4.2: List any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element?

. . . .
. . . .

Please record the Start and End times (using the following format: hh:mn:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy [ ] Easy [ ] Neither easy nor difficult [ ] Difficult [ ] Very Difficult

Q5: [Fig. 10] We plan modify the responsibility ***EditEventForVisit*** in **Process Map** (within Observer component).

Q5.1 Please color the path and draw a ***circle*** on all the elements that might be impacted by modifying this responsibility.

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy [ ] Easy [ ] Neither easy nor difficult [ ] Difficult [ ] Very Difficult

Q5.2: Please list any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element?

. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy [ ] Easy [ ] Neither easy nor difficult [ ] Difficult [ ] Very Difficult

Q6: [Fig. 10] We plan to add a new responsibility between ***ScoreEvent*** and ***AEMS-StoreReview*** in **Process Map** (within Reviewer component).

Q6.1: Will the responsibility ***WarnReviewer*** be impacted?

Yes [ ] No [ ]

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [....................] End time: [....................]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q6.2: Please list any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element?
. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [....................] End time: [....................]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Part 2.2: Comprehension of URN model using jUCMNav CIA feature.

Consider the following URN model of **Commuting system**. [Commuting]**. See Figures 12-20.**

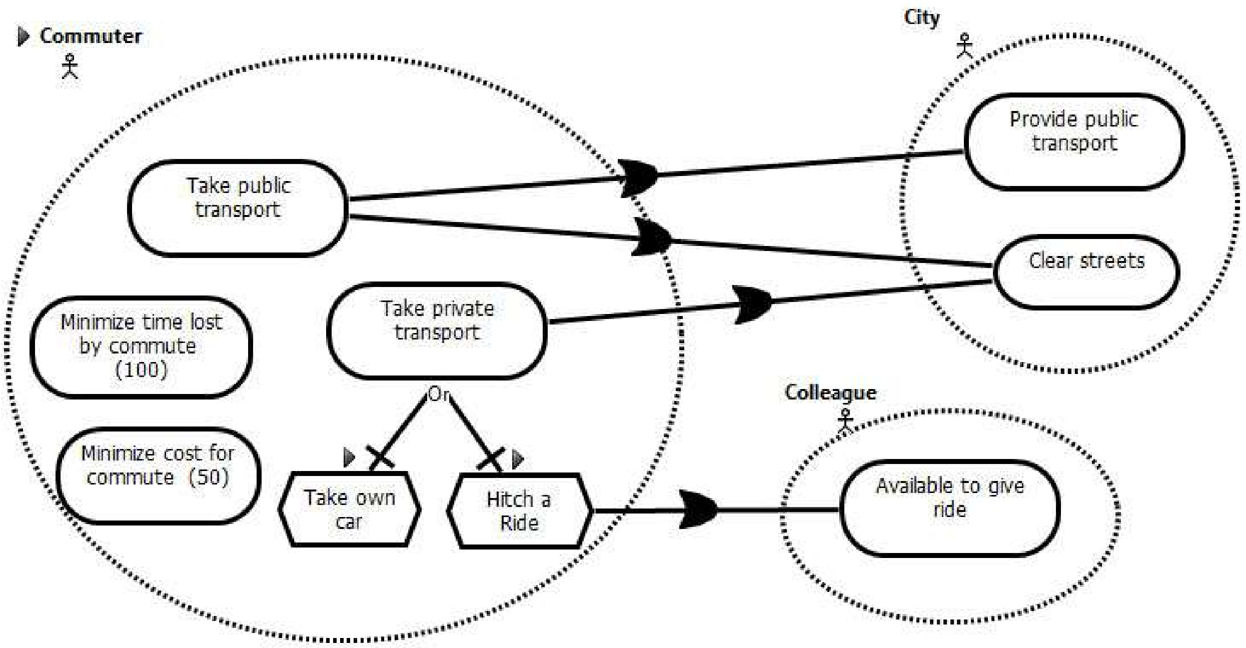


Figure 12 "Stakeholders" Map – GRL

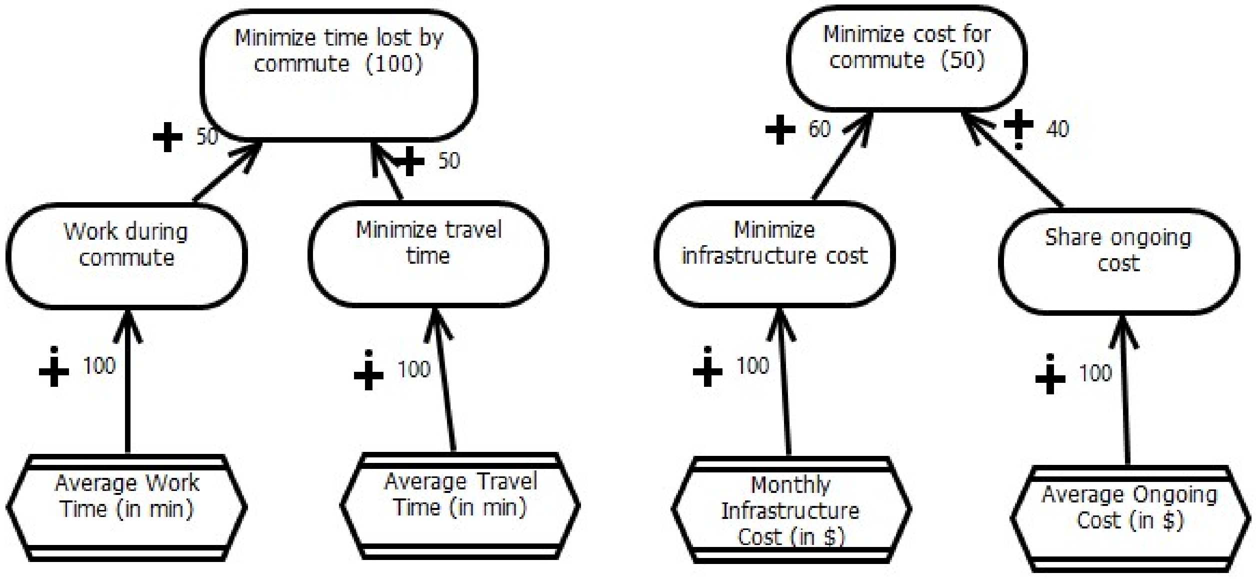

Minimize time lost by commute (100)
+ 50
+ 50
Work during commute
Minimize travel time
+ 100
+ 100
Average Work Time (in min)
Average Travel Time (in min)
Minimize cost for commute (50)
+ 60
+ 40
Minimize infrastructure cost
Share ongoing cost
+ 100
+ 100
Monthly Infrastructure Cost (in $)
Average Ongoing Cost (in $)

Figure 13 "Commuting-Reasons" Map - GRL

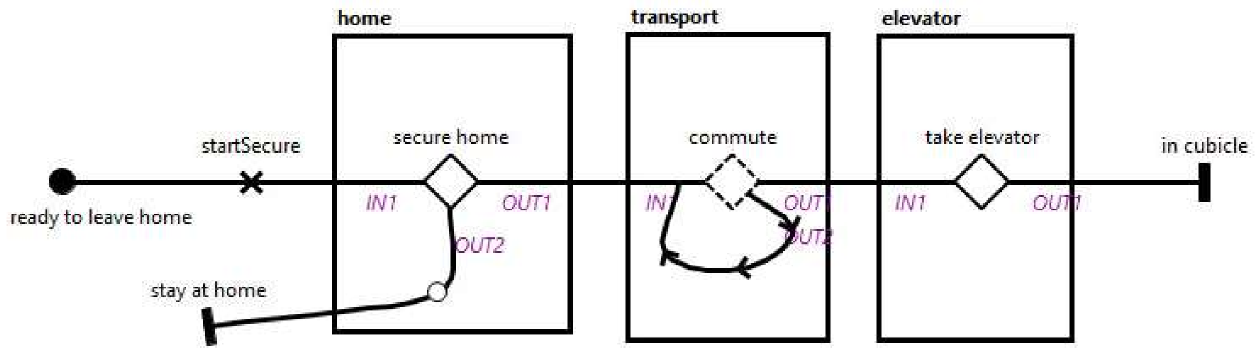


Figure 14 "Commuting Map" – UCM

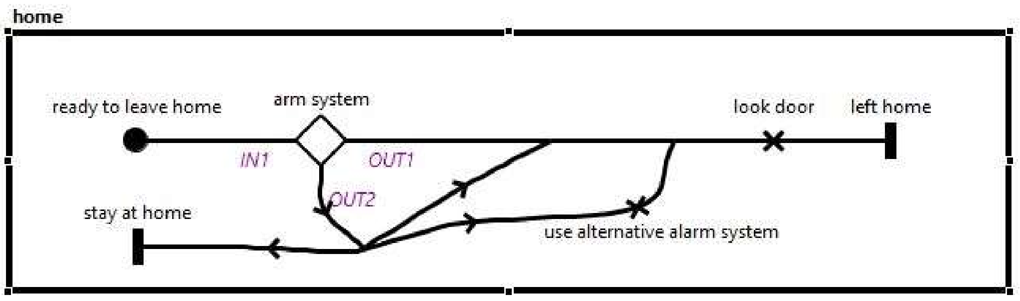


Figure 15 "Secure Home" plugin - UCM

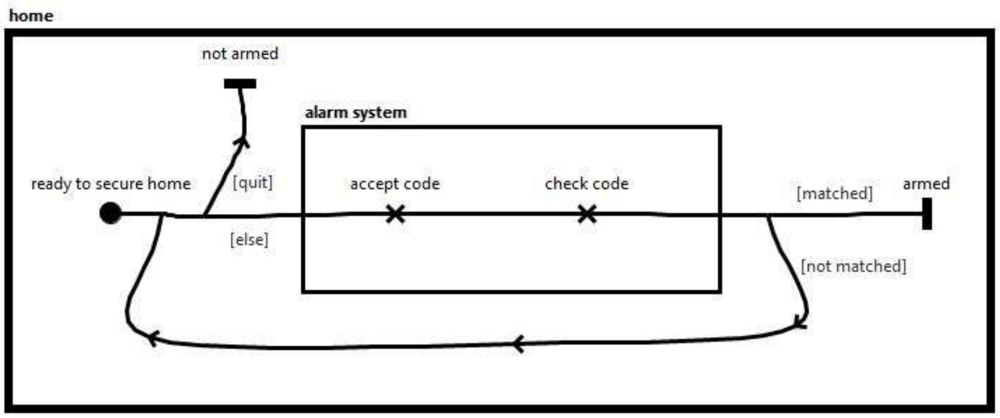


Figure 16 "Arm System" plugin – UCM

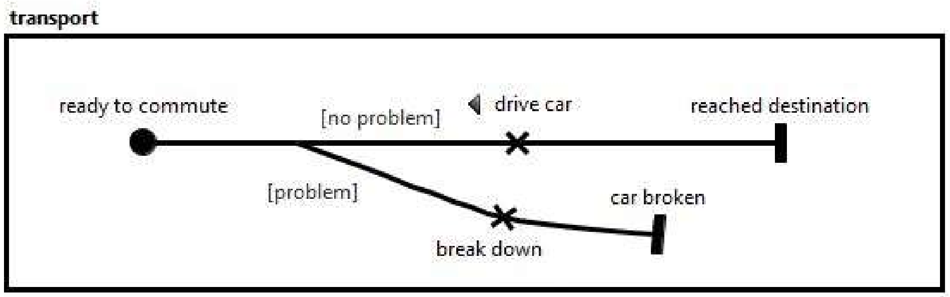


Figure 17 "Car" plugin - UCM

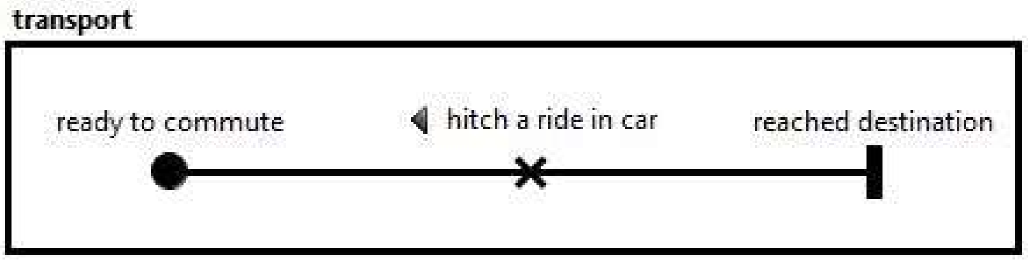


Figure 18 "Hitch a Ride" plugin - UCM

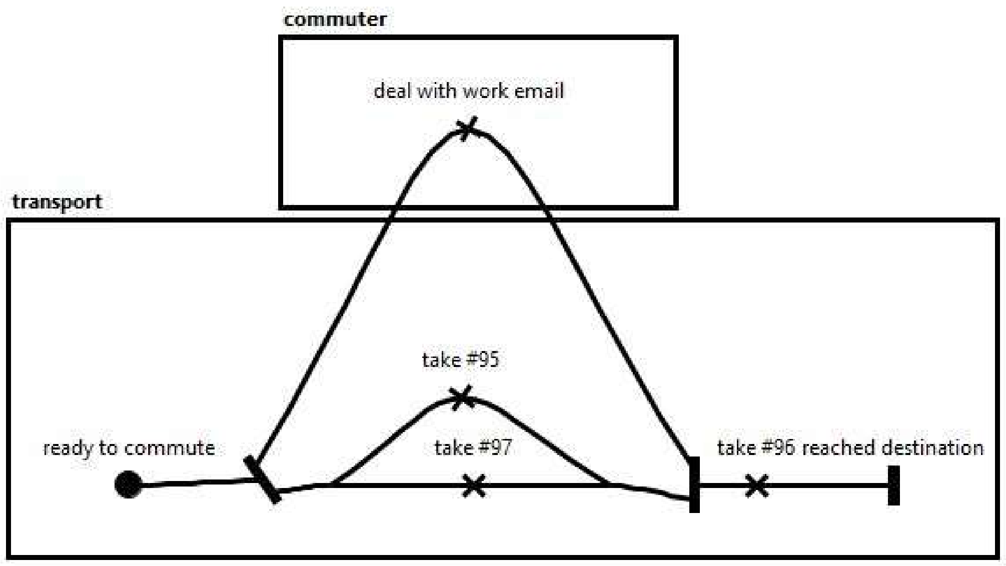


Figure 19 "Reqular Bus" plugin - UCM

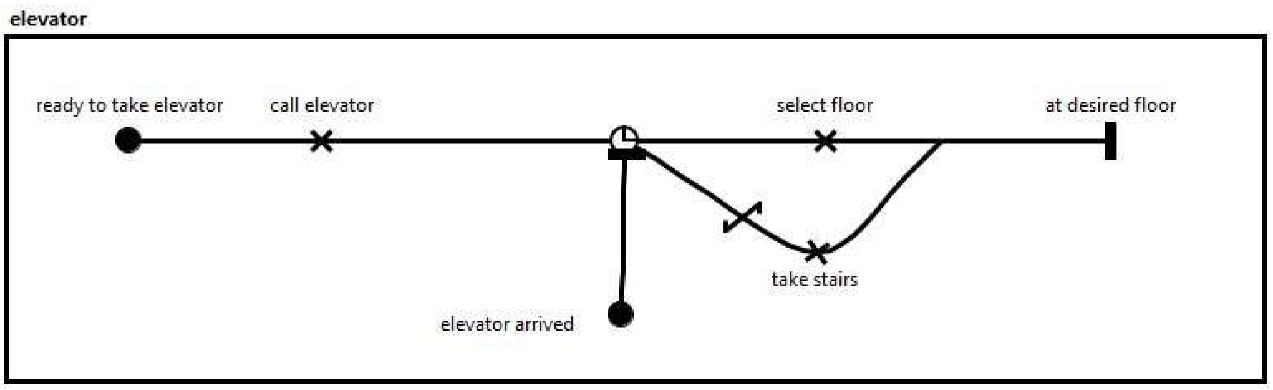


Figure 20 "Take Elevator" plugin – UCM

Please answer the following questions and note the time spent in answering each question:

Q1: [**Fig. 12**] We plan to delete the GRL Goal "***Available to give ride"*** *in* ***"Stakeholders map".***

Q1.1: Please mark, on the model, all elements and links that might be impacted by this modification. (Use a different color):

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [....................]   End time: [....................]

Please check the level of difficulty of performing the task:

[ ] Very Easy   [ ] Easy   [ ] Neither easy nor difficult   [ ] Difficult   [ ] Very Difficult

Q1.2: List any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element.

. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [....................]   End time: [....................]

Please check the level of difficulty of performing the task:

[ ] Very Easy   [ ] Easy   [ ] Neither easy nor difficult   [ ] Difficult   [ ] Very Difficult

Q2: [**Fig. 12**] We plan to modify (i.e. replace the **Or** by **And** decomposition), then, apply CIA to the ***Take private transport*** in ***Stakeholders map***.
Q2.1: Please mark, on the model, all elements and links that might be impacted by this modification. (Use a different color)

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [....................]   End time: [....................]

Please check the level of difficulty of performing the task:

[ ] Very Easy   [ ] Easy   [ ] Neither easy nor difficult   [ ] Difficult   [ ] Very Difficult

Q2.2: List any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element.

. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [………………….] End time: [………………….]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q3: [**Fig. 13**] We plan to add a new **Indicator** and a new positive **contribution link** to "***Work during commute"*** in **Commuting-Reasons map**. will the goal ***Minimize travel time*** be impacted?

Yes [ ] No [ ]

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [………………….] End time: [………………….]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q4: [**Fig. 1**8] We plan to add a new responsibility between ***hitch a ride in car*** and ***reached destination*** in **Hitch a Ride Map (**within transport component**)**.

Q4.1: Please color the path and draw a ***circle*** on all the elements that might be impacted by modifying this responsibility.

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: [………………….] End time: [………………….]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q4.2: Please list any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element?

. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q5: [**Fig. 18**] We plan to delete the responsibility ***hitch a ride in car*** in **Hitch a Ride map** (within transport component).

Q5.1: Please draw a ***circle*** on all the elements that might be impacted by deleting this responsibility?

Please record the Start and End times (using the following format: hh:mm:ss):

Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q5.2: List any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element?
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mn:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q6: [**Fig. 15**] We plan to modify the responsibility ***look door*** in **Secure Home map** (within home component).

Q6.1: Will the responsibility ***startSecure*** in **Commuting map** be **impacted?**

Yes [ ] No [ ]

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult

Q6.2: Please list any impacted URN Links, if any, as a result of this modification. Please indicate both the URN start element and the target element?

. . . .
. . . .
. . . .

Please record the Start and End times (using the following format: hh:mm:ss):
Start time: […………………] End time: […………………]

Please check the level of difficulty of performing the task:

[ ] Very Easy	[ ] Easy	[ ] Neither easy nor difficult	[ ] Difficult	[ ] Very Difficult