A Rule-Based Framework for Supporting Automated Change Impact Analysis in the Cancer Registry of Norway

Thomas Schwitalla

Master Thesis
Programming and Networks
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

May / 2017

A Rule-Based Framework for Supporting Automated Change Impact Analysis in the Cancer Registry of Norway

2017

A Rule-Based Framework for Supporting Automated Change Impact Analysis in the Cancer Registry of Norway

Thomas Schwitalla

http://www.duo.uio.no/

Trykk: Reprosentralen, Universitetet i Oslo

Abstract

This thesis investigates the issue of change impact analysis in a special case of health information system: the Cancer Registry of Norway. The Cancer Registry of Norway (CRN) is an institution dedicated to the population-based research on cancer. When gathering cancer data from medical institutions obligated to report their results of treating their patients, the CRN must ensure the validity and correctness of the reports to maintain a high standard of data quality. The data and the cancer data quality is however constantly changing as more information on a patient is recorded as he or she are subjected to more diagnostic and therapeutic procedures as their cancer disease develops. The data might be compromised by erroneously entered values in cancer reports or by inconsistences between cancer reports. Cancer data might also change because of increasing precision, as more information is recorded over time. To keep up with the effects of these changes, the CRN must frequently repeat its data quality measures, often manually. Furthermore, more complex types of changes occur within the CRN as well. Due to the ever-evolving nature of medical science, the need to change the schemas recording the cancer data within the registry databases can arise as well. These changes can accompany the need to change the business rules underlying the data registering process, known as cancer coding rules. When these rules change, persistent data constrained and calculated by them change their states; what was previously valid may now be invalid, and values may be recalculated to unpredictable results. Identifying the extent of these side effects is currently a manual, time-consuming, and error-prone process. Hence, the CRN has identified a strong need for automated change impact analysis in their domain. However, they have found existing commercial off-the-shelf solutions lacking due to the specificity of their problem. Likewise, the research that concerns the change impact analysis of software systems does not appear to have considered these specific problems within a similar domain.

To address the above-mentioned challenges, this thesis proposes a framework for automated change impact analysis in the Norwegian cancer registry domain, inspired by more generic approaches from the field of software change impact analysis. Our approach accurately analyzes changes based on the type of object that is changed, and the type of change that is proposed to the given object according to domain-specific impact analysis rules. We evaluate our framework in a case study with the CRN, using realistic changes and cancer data sets, and real cancer coding rules. The results of the evaluation show that our framework can

automatically analyze the given changes (with various change types) and provide corresponding impact output compliant with the medical domain knowledege of the CRN. The further work of the approach lies mainly in evolving the current change impact analysis rule set so as to fully support all cases of changes and impacts.

Foreword

I extend my thanks to my family, who would support me to the end despite not knowing exactly what it is that I am doing (except my two nephews, who will one day be better than me at everything). Furthermore, I respectfully thank my supervisor, Shuai, with Simula Research Laboratory. He always made sure that I was never just “correct”, but also “accurate”, and always kept an open mind to my ideas. The near-unlimited supply of coffee, fruit, and lunches provided by his organization was also of great help. Finally, special thanks to the Cancer Registry of Norway are in order. They provided us with a unique and fun challenge, and supported every aspect of our work as far as they could.

Table of Contents

1 Introduction

The changes to software systems are inevitable, continual, and carry potentially costly consequences. As the system evolves, and later during the maintenance phase of its lifecycle, changes in the system requirements specifications will invariably cause unintended side effects. These effects can include the introduction of new defects that cancel out the intended purpose of the changes, leading to undesirable behavior, reduced stability or performance. The later the changes occur, the greater becomes their potential *impacts*, and the greater becomes the potential need for rework. It has been estimated that more than 50% of maintenance costs come from changing software [1]. Even small changes can spread, "ripple", to a great number of components and units in the system [2], and manually assessing the set of affected software artifacts is a daunting task even for the most experienced engineers. Hence, tool-supported automated *software change impact analysis* (CIA) has become an increasingly desirable asset in software engineering; it is likely to provide considerably greater precision and speed of assessing the potential ramifications of changes than that of any manual, human analysis technique [3].

The problem of semi-automated and automated software CIA is complex but nonetheless well-studied—since the 1970s hundreds of approaches have been proposed [4, 5]. Recent reviews of works in this field have found approaches that have been proposed for any programming or software development paradigm, for a multitude of application domains, using analysis techniques that range from naïve search to novel use of machine learning algorithms. Approaches have even been designed that can analyze change impact across more than just the software source code itself, including the system requirements and specification artifacts [6-8].

One application domain that may be experiencing a great a need for automated CIA as any domain, but have specific needs that seemingly no single approach as of yet has been designed to manage, is medical information registries. One such example is found in Oslo, Norway, in the Cancer Registry of Norway (CRN). Here, the relevant changes occur to (1) persistent cancer data that has been gathered from medical institutions that treat cancer; (2) their databases schemas; and (3) rules known as *cancer coding rules*, which ensure the quality of the gathered data by validating them, and aggregate them into a unified format for all types of cancer data sources (which are validated by their own set of rules). As data, *cancer*

messages, that come in from the various sources continuously as a patient are subjected to different diagnostic procedures and treatments, coding rules validate these data to hinder erroneous data to enter into the system. The main unit of data that is curated by the CRN, the *cancer case*, is dependent on these messages since the cancer case data is calculated based on the cancer coding rules from the values in one or several of the message fields. Changes might thus occur to cancer cases on a daily basis as the cancer messages are submitted to the Cancer Registry. Therefore, a web of dependencies exists between the persistent cancer data objects, but the real complexity lies in the vast set of cancer coding rules that constrain and calculate these objects. Numerous dependencies exist between cancer data objects and cancer coding rules, in both directions.

1.1 The CIA Problem in the CRN

As the CRN is concerned with the research on cancer, data quality is of the utmost importance. When changes to the cancer data occur, every step in their data registering procedure must be repeated to ensure that the cancer data that have been already validated is still valid. When changes to the database schemas and the cancer coding rules must be made due to advancements in knowledge of cancer treatment, thorough change impact analysis must be done because the potentially affected data sets are larger and can include other cancer coding rules (which may also need to be changed, leading to further change ripple). With the current practice, the estimation of what is expected to be affected of a certain change and what is not to be affected is an entirely manual process. Potentially, the associates of the CRN work according to a non-uniform set of unwritten analysis rules and heuristics when making these estimations. Aside from this process being tedious and time-consuming, it is also error-prone, potentially compromising the quality of the cancer data and thus the research conducted by the CRN. Therefore, there is clearly a great need for automated CIA in the CRN; but it must provide sufficiently fast and precise estimation.

1.2 Scope and Contribution

We aim to improve the current practice of the CRN by designing and developing a framework for an automated CIA approach specifically for application in their medical domain. As, to our knowledge, no single software CIA approach has been proposed for the specific domain of cancer information registries, *and* for the particular types of changes that occur in the CRN,

we believe that our contribution is novel. Likewise, we have to draw upon works from other fields, and the field of general software CIA appears to us as the richest source of inspiration. An early step in designing our approach is to review the state of the art of this field, and to identify key works that can be built upon. Our approach may or may not have implications to the field of general software CIA, but most likely it will stand as an example on a domain-specific CIA approach in the field of health informatics.

Our proposed approach will be implemented in a simple prototype that we use to assess the applicability of the approach within the CRN. *Applicability* refers to whether the approach can (1) correctly identify and define the various types of changes that occur in the CRN; (2) correctly represent the various objects that exist in the CRN domain; (3) simulate the defined changes on the domain object, and produce correct estimations of what should (or should *not*) be affected given some change; (4) correctly describe the consequences to each object identified as affected; and (5) scale within reasonable time and computing resource limits. *Correctness* is with regards to what is expected by the CRN stakeholders. An important part of our work must then be to first study the CRN domain in order to identify and classify change types and the concepts on which they can be applied. Furthermore, efforts must be made to uncover the rules by which the current practice of manual CIA is performed. Finally, having designed and implemented the approach, we iteratively evaluate its applicability using both qualitative and quantitative methods. To summarize, we state our main goal as follows:

> *To propose an efficient approach with user-friendly tool support for performing automated change impact analysis in the context of the Cancer Registry of Norway.*

1.3 Structure of the Thesis

1.3.1 Chapter 2: Background

In chapter 2 we establish the theoretical background of this thesis. We begin in section 2.1 by examining the established definitions of the term *software CIA* to see what is typically required and expected of any CIA approach. Here, we introduce key terms that will be used throughout the thesis when talking about CIA approaches, including our own proposal. Additionally, we give an overview of various types of CIA approaches that have been identified by other authors. In section 2.2 we investigate the context and current practice of the CRN, with the purpose of understanding the problem with which we aim to assist. This

investigation is aided by previous works that has studied the CRN. Finally, in section 2.3, in light of the knowledge acquired from the domain study we examine select works from the field of software CIA in order to find techniques that could be used in solving, or adapted to, the CIA problem in the CRN.

1.3.2 Chapter 3: The Approach Framework for CIA in the CRN

In chapter 3 we present the approach to which we propose to solve the CRN CIA problem. We start with giving the high-level design of the approach framework, discussing its requirements and design features and relating them to the CRN problem. Our conceptual approach framework comes in three parts, and to each of them we dedicate a section that explains it (sections 3.2, 3.3, 3.4, and 3.5). Concluding chapter 3, section 3.6 presents the prototype software tool that implements the approach framework for the purpose of evaluating it empirically in cooperation with the CRN.

1.3.3 Chapter 4: Evaluation of the Approach

In chapter 4 we explain our methods for evaluating the proposed CIA approach. We evaluate in two stages: (section 4.1) iterative qualitative assessment of the applicability of the framework and the usability of the prototype and (section 4.2) quantitative performance measurements using well-established metrics. For each evaluation stage, the respective section describes the research questions, methods, and results. Chapter 5 discusses the results of the evaluation. Chapter 6 concludes our work by relating our findings to our main goal, and comparing our proposed approach with the related works discussed in section 2.3.

2 Background

2.1 Software Change Impact Analysis

The most basic definition of the concept *change impact analysis* in the field of software engineering is [2]:

> *[…] the activity of identifying what to modify to accomplish a change, or of identifying the potential consequences of a change.*

The field of software CIA concerns the research and development of techniques and *approaches* for determining the effects of changes to *software artifacts*. A CIA approach can include automatic or semi-automatic software tools, or manual techniques. The software artifacts are not only source code elements of the software itself; the term has in numerous approaches been extended to also cover artifacts of the engineering *process*. This includes system models (e.g. in Unified Modeling Language, UML) [6], [8], requirements [7], and even business rules of the domain in which the software is deployed [9]. The goal of any CIA approach is to identify *ripple effects* of changes in order to predict further *side effects* that go beyond the initially changed software artifact. Ripple effects are effects caused by making a small change to a system which affects many other parts of the system [10]; side effects are errors or other undesirable behavior that occurs as a result of the change [11]. The crucial term *impact* (noun) refers to the artifacts that are determined to be affected by a change, and therefore worthy of inspection [2] [1].

As an example CIA scenario in which we apply the aforementioned terms, consider a change to a source code artifact: a function (its semantics and implementation are immaterial). The *change request* here is the function and the change to which it is subjected. Changing this function to, say, return a different type of value may affect any other functions that call the changed method and rely on its return value. This change may likely have been the consequence of a change to the requirements of the function—in other words, the requirement change rippled further to its implementation. The side effects of the source code change could be that the other functions that rely on the changed function no longer run correctly as they expected a different value, thus they would themselves need to be changed (or the original

[1] Note that in our approach, given in chapter 3, the term "impact" is used in the sense of "what is the impact of change *c* to object *o*".

change may have to be reconsidered). In this scenario, the impacts are the functions and the requirement (other impacts undoubtedly exist).

2.1.1 The General CIA Process

Bohner and Arnold first attempted in 1993 to define CIA and to classify the various existing CIA approaches [2]. The authors state that to accomplish CIA, three things are required: (1) a proposed change; (2) an object to change; and (3) a way to estimate what must be done to do the change. In their *impact analysis framework*, the generic CIA process is modeled after a basic manual CIA approach, with which any software engineer will be familiar. First, some source code artifact is proposed for change (possibly by a requirements engineer or someone else who was not the original developer). Then, the original or new developer will be approached with the proposed change, and asked to provide an estimation of what must be done to implement the change. This estimation should include not only changing the requested artifact, but also what other artifacts must be examined in case they are affected in some way. Also, "affected in some way" should be defined—it is usually useful to know what would happen to the artifacts. The programmer responsible for the CIA would then have to tap into their knowledge of the system, perhaps needing to consult associates in order to fill any gaps. During this step, several *impact sets* will be computed in an iterative manner. The first of these sets, which is the superset of all further impact sets, is the *starting impact set* (SIS). The SIS should include all artifacts in the system that *could* be (not necessarily *is*) affected by the given change. Then, the analyzer must somehow estimate what artifacts in the SIS that is (probably) impacted; this is the *estimated impact set* (EIS). The EIS computation would require some structured procedure that simulates the application of the change on the SIS. In an automated approach, this would imply that some algorithm is implemented. Of course, the human programmer could manually follow an algorithm, but is likely to spend a great deal more time than a computer doing this (and more likely to make mistakes in the process).

The EIS is an important concept, as any CIA approach can only aspire to produce estimations; exhaustive coverage of change impacts is unfeasible in all but the smallest cases, much in the same way exhaustive software testing is unfeasible. The set that would be produced given a hypothetical perfect CIA solution would be an EIS that is identical to the *actual impact set* (AIS), which includes only artifacts that are impacts. A widely used method of evaluating

CIA approaches is to measure its performance in terms of *precision* and *recall* of its EIS when compared to a manually crafted AIS[2] [4, 5].Precision measures the fraction of objects that were determined to be affected that should actually be affected, and recall measures the fraction of actually impacted objects found, out of all objects in the AIS (formal definitions of precision and recall are given in section 4.2.3). Thus, performance of CIA approaches is usually defined in terms of the relationships of the *system* (the set of all objects in the system on which CIA is performed), the SIS, EIS, and the AIS. Ideally, the AIS is always overlapping with the EIS to some degree, and the EIS should be a minimal subset of the SIS.

2.1.2 Functional Parts of CIA Approaches

According to the theoretical CIA approach framework of Bohner and Arnold, to express a specific change, each CIA approach has its own *interface object model* of the artifact objects that can be captured in the application domain and their relationships (or dependencies) to one another. In other words, the artifact object model is the way in which the CIA approach understands its application domain. The objects and relationships can be predefined or user-defined, or both. The approach provides an interface to this model, so that the input (expressed in the interface object model) can be translated to the *internal object model* of the approach. The internal model, which captures what objects and relationships the approach uses to accomplish CIA, is usually stored in some repository, which enables loading, browsing, and modifying objects and relationships.

The *impact model* of the CIA approach defines "what affects what". This can be given by rules and algorithms that determine when a change to an artifact object will affect another object. The impact model is implemented by the *tracing/impact approach*, which defines how objects and dependencies are represented, how impact rules are implemented, and the specific search algorithms used to find impacted objects and dependencies.

2.1.3 Classes of CIA Approaches

The classic impact analysis approach framework of Bohner and Arnold is useful for talking about the components of CIA approaches. We adopt this framework when categorizing the parts of the proposed CIA approach for the CRN in chapter 3. However, the framework does

[2] In the context of precision and recall measurements, some readers may be more familiar with the term *gold standard*, rather than actual impact set.

not provide a taxonomy of CIA approaches, which makes it difficult to use for comparing them by their more general characteristics. When classifying and comparing a set of CIA works related to our approach in section 2.3, we apply a more recent framework: that of Steffen Lehnert from 2011[4]. As stated in his proposal for this framework of comparison of CIA approaches, the framework of Bohner and Arnold has since 1993 not been extensively applied for comparing CIA approaches. Lehnert also laments that Bohner and Arnold's evaluation of their framework is somewhat limited in size and scope. Lehnert's framework is based on a comprehensive review of 150 semi-automated and automated CIA approaches (starting with reviews of existing CIA approach frameworks), and is empirically evaluated by checking the coverage of each criterion defined in the framework.

Lehnert's taxonomy encompasses three different scopes of interest: (1) *source code analysis*, which is either *static*, *dynamic* or *online*; (2) *formal models*, which are further divided into architectural and requirements models; and (3) *miscellaneous artifacts*, which include a wide range of documents and data sources.

Source Code Analysis

In these approaches, source code files, class packages, classes, methods, statements, and variables are analyzed to predict the propagation of changes. Such techniques have the fundamental requirement of existing source code on which to be applied. Furthermore, their application is often restricted to programmers due to their technical nature. Static code analysis "extracts facts from source code to build call graphs, *slices*, and other representations which are used to assess the impacts of a change". Slices are sets of program statements that may affect values of some variables at some point in time. Dynamic and online approaches instrument the code or compiled binaries to collect information about method executions (*execution traces*). Dynamic analysis is when the execution traces are analyzed after program execution; online analysis is when they are analyzed at runtime. This is by far the largest scope (encompassing 65% of all the reviewed literature); due to the large amount of studies found, the review of this scope is divided by the various techniques identified:

Call Graphs: This is the analysis of the call-behavior of a system to assess the impact of a change. Source code is analyzed statically while method or function calls are extracted and stored in a graph or matrix, which is used to estimate the *change propagation* (the ripple of a change). "Traditional" call graph-based approaches are plagued by low precision due to the

need for limiting the propagation to a tractable level; many CIA works have proposed methods in which to deal with this issue.

Dependency Analysis: A variety of dependencies between source code artifacts exist, e.g., control data or inheritance. As with call graph approaches, dependencies are extracted from static code analysis and stored in a graph or matrix. The object-oriented (OO) paradigm results in a great number of dependencies which complicates this type of approach; numerous techniques have been proposed to address this.

Program Slicing builds upon dependency analysis, and removes all program statements that are not related to the slicing criterion, i.e. statements that do not affect the state of a variable and thereby being of no use for impact analysis. Static slicing has been shown to be computationally expensive and thus various approaches spend great effort on optimizing it.

Execution Traces: Unlike static call graphs, dynamic execution traces contain only those methods that have been called during the execution of a program. Similarly to static call graphs, however, they perform CIA by analyzing which methods were called after the changed one. This type of approach was established to overcome the limitations of static slicing and call graphs.

Explicit Rules: Design, domain, and expert knowledge can be used to form strict impact rules, which determine which artifacts have to change if a certain artifact changes. Such rule-based approaches often rely on a classification of the various changes that can occur in the application domain and different relations between artifacts.

Information Retrieval approaches infer relations between similar terms in different documents. Example techniques are *vector space models* and *probabilistic models*.

Probabilistic Models are based on well-established mathematical models and theorems, and compute the probability of an artifact being impacted by a change. Historical information about changes is required by such approaches.

Note that, the source code analysis techniques can be, and have been, applied under other scopes as well.

Formal Models

The key motivation of architectural model-based approaches is that they enable IA earlier in the development. This is because as they do not require source code or historical information. Furthermore, they also enable CIA in finer levels of granularity later on in the development process, depending on the underlying modeling language. An additional advantage of model-based approaches is that they may be useful to other stakeholders than programmers (e.g., requirements engineers).

If system requirements are encoded in formal modeling languages, they can be analyzed for change impact. This is naturally useful as requirements undergo frequent changes over the course of a system lifecycle. When requirements are expressed in plain text however, natural language processing and information retrieval approaches must be used to perform CIA.

Miscellaneous Files and Artifacts

Various artifacts ranging beyond source code, models, and requirements can be changed, causing effects to software. As with other scopes, methods using historical data on changes to artifacts are prevalent. Probabilistic approaches have also been applied under this scope of interest.

Combined Scopes

Most of the approaches reviewed under this scope are concerned with bridging the gaps between source code, requirements, and/or system models. Some approaches also focus on test case selection, in particular for regression testing purposes.

2.2 The Domain and Current Practice of the Cancer Registry of Norway

This section gives insight into the CRN as seen through the lens of the research project “An Innovative Approach for Longstanding Development and Maintenance of the Automated Cancer Registry System”, which is collaboration between the CRN and Simula Research Laboratory. This project aims to “develop systematic approaches to facilitate maintenance of the cancer registry system” [12]. Note that for the sake of brevity, the project will be henceforth referred to as the *MBE4CR* (Model-Based Engineering for Cancer Registry). One

important step to designing the proposed CIA approach was to gain knowledge of the CRN domain by reviewing the works produced so far by the MBE4CR. They proved crucial for understanding the problems faced by the CRN, and thus provided a basis for the requirements of the approach.

First, the core business process in the CRN, the *cancer data registering process*, is explained. This explanation is assisted by a conceptual framework of the CRN domain that has been defined by the MBE4CR [12]. Second, the various challenges of the CRN are described, along with an introduction to the *Model-Based Framework for Cancer Registry* (MBF4CR), a software platform developed by the project that implements proposed solutions to these challenges [13].

2.2.1 The Cancer Registry of Norway

The CRN is an institute dedicated to the population-based research on cancer [14]. Located in Oslo, CRN is a part of the South-Eastern Norway Health Authority, organized as an independent institution under Oslo University Hospital. One of the CRN's primary tasks is to gather *cancer messages* from various medical *sources*, which include any medical entity involved with the medical review, treatment, and monitoring of cancer patients in the country. These messages are *aggregated* into *cancer cases*, which form the data for the statistical analysis of national cancer incidence and prevalence. Since 1953, the CRN has annually published its report, *Cancer in Norway* (CiN), on these statistics. CiN contains information such as the incidence rate of cancer in the population, mortality and rate of mortality, prevalence, and relative survival rate. In addition to this publication, the CRN also provides custom data for media, municipal health services, hospitals, research projects, and governmental institutions.

2.2.2 The Registering Process

Viewed on a deeper level of abstraction, the CRN's process of gathering cancer data requires several steps from the moment a message is reported from a medical entity: (1) *Cancer Message Validation*: the message must first be *validated* according to a set of *cancer coding rules;* (2) *Cancer Message Aggregation*: the message, if valid, is *aggregated* into a *cancer case*; and (3) *Cancer Case Validation*: the updated cancer case that resulted from the aggregation must itself be validated because errors may have been introduced during

aggregation process [12]. This process implies a system of some complexity; each step is in itself a process involving a vast set of rules and numerous cases that must be handled appropriately depending on a variety of outcomes. To provide a basic-level understanding of the overall data processing done by this *automated cancer registry system* (ACRS), this section will explain each of the key concepts from the domain of the CRN. These concepts are (1) *cancer messages*, (2) *cancer cases*, (3) *patient history*, and (4) *cancer coding rules*. As the proposed CIA approach must incorporate these concepts into its own artifact model, a study of them is necessary.

Cancer Messages

From the perspective of, say, a clinician, a cancer message is a form containing information that has been gathered during an examination of a patient (e.g., through methods such as x-ray or magnetic resonance imaging), and information about the patient. A radiology lab is one example of a source of cancer messages. Other sources can be pathology departments, specialists and general practitioners, and morgues. These entities are all obligated to report to the CRN about all cases of cancer that they record. This includes early stages of cancer and benign tumors [15]. The sources report their messages mainly through the CRN's Electronic Messaging Service (KREMT – Kreftregisterets Elektroniske Meldetjeneste) [15]. KREMT provides both the possibility to submit extensible markup language (XML) files by electronic data interchange, and to report through a web-based solution that is available to all entities connected to The Norwegian Health Network. Other ways of reporting messages are allowed, as long as they conform to the messaging specifications of the CRN, and the messages can be recognized by their system.

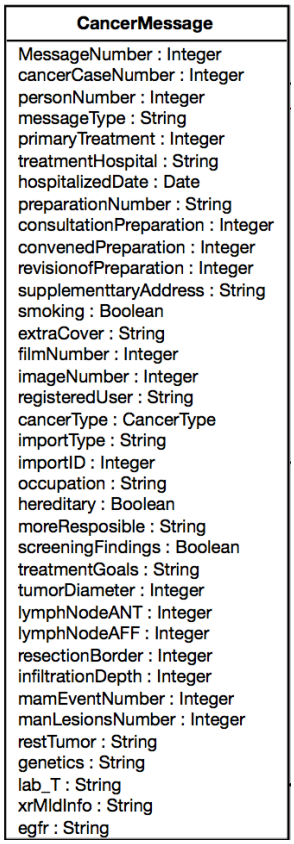

Figure 2-1: An excerpt of *CancerMessage* Class.

The specifics of what a given cancer message contains naturally depend on the source of the message (e.g. diagnostics, treatment, or follow-up) and on the type of cancer; some types of cancer even have their specific forms.

In the MBE4CR project, domain knowledge is captured through the use of Unified Modeling Language (UML) *class diagrams* [16]. Class diagrams show the

structure of a system, in this case the CRN domain, at the level of *classes*. One class identifies a set of objects that share the same features, constraints, and semantics (meanings). The top box of the class gives its name, and the following two gives *attributes* and *operations*, respectively. Attributes are the structural features of the class, and operations are its behavioral features [17]. Figure 1 shows a UML class of the *CancerMessage* class, containing a subset of the fields that may be recorded by the system for each incoming message [12]. The fields make up the attributes of the class, which in the case of cancer messages correspond to the database schema of the cancer message form. There are no operations in this class—it simply defines a form and can have no behavior, only structure.

At the implementation level of the CRN ACRS, a *CancerMessage* records fields containing source (*MessageType*) and cancer type (*CancerType*), in addition to fields for the various possible details about diagnostic procedures and results, as well as different treatments given to the patients. In total a *CancerMessage* has 64 fields currently, some of which may be uninitialized (they contain no values, referred to as *null values*) as different sources report different information, e.g., a message from the radiology lab would provide values for fields related to an X-ray screening, but not for fields related to, say, surgery.

2.2.3 Cancer Case and Patient History

A *CancerCase* in the CRN database represents a single case of one type of cancer for one specific patient. The patient, represented by their *PatientHistory*, may suffer from more than one type of cancer at the same time. Hence, several cases can be associated with the same patient and the *PatientHistory* thus refers to a set of cancer cases. The cancer case is the main unit of data on which the CRN bases its statistics. The values of the fields of cancer cases, unlike the fields of cancer messages, are not set by the hands of some medical associate at the source of the message, but are set during the aggregation process at the CRN (section 2.2.4).

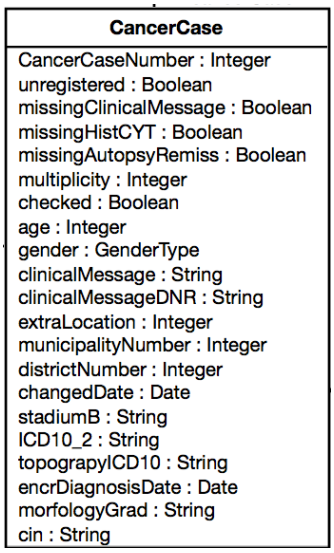

Figure 2-2: An excerpt of the *CancerCase* Class.

Figure 2-2: An excerpt of the *CancerCase* Class, again from [12], shows the CancerCase class. Note that the class shown here does not capture all possible fields recorded for an actual cancer case. As cancer cases share some data with their related cancer messages, this requires a class of its

own: the *CommonFields*. The shared data include certainty of diagnosis, date of registration, various state variables, and of course details of the cancer. Figure 2-3 shows the CommonFields class.

2.2.4 Cancer Coding Rules—Aggregation and Validation

Cancer messages need to be checked for *validity* as the nature of their contents is highly safety-critical; they are the personal information of a patient along with the various data related to their session at the medical source that reported the message. The lab technicians, doctors, and other practitioners at the various message sources, human as they are, must be expected to make the occasional error when entering data into the forms. Furthermore, after aggregation, the resulting cancer case may contain new errors for which a check must also be done.

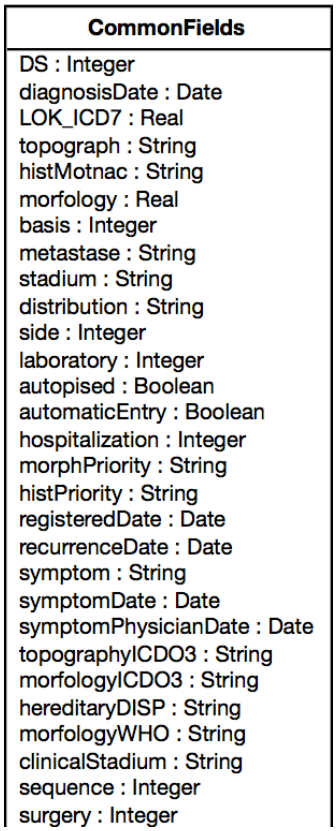

Figure 2-3: The CommonFields class, which represents shared fields for cancer cases and cancer messages.

What is "valid" must be defined by some rules, and what is an "error" must then be a violation to any of those rules. In the CRN a set of about one thousand cancer coding rules exists, according to which the two discrete processes of validation and aggregation are carried out. This rule set can be viewed as two subsets: *Validation Rules* (VARs) and *Message Aggregation Rules* (MARs) [12]. The validation rules are used for validating fields in a cancer message or a cancer case; these rules can thus be further classified into *Message Validation Rules* (MVRs) and *Case Validation Rules* (CCVRs). The MVRs determine if a message is eligible for aggregation, but a part of the MARs responsibility is to make the final decision on whether or not the validated message will be included aggregation. Note that many rules are shared by each of the validation steps—this implies a third subtype of validation rules, which will henceforth be referred to as *Common Field Validation Rules* (CFVRs).

Cancer Message Validation

The first line of validation is done by checking the fields of the cancer message, sequentially and exhaustively, against the message validation rules. At the most basic level these rules

define valid entries for fields such as age and gender. An example of a basic rule is thus “The value of field *age* must be greater than 0 and smaller than 120”. If this condition is shown to be true, the system will proceed with checking the next field of the cancer message. An example of a more complex rule might check several fields:

morfologi = 1527/39 and messageType = K, R require topografi = 44.x

Note that the above example shows the way rules are actually written in the rule tables of the CRN. This is an MVR; as one or more of the fields it constrains exist in the CancerMessage schema only (i.e., it is not a common field). Understanding the exact purpose of this rule and what the values represent requires domain knowledge that is beyond the scope of this chapter. What is important here is that this is an example of a rule that requires consistency between fields. A check against this rule will only pass if all of the *required* values fall within their required ranges, or if the values of the field of the cancer message on the left-hand side of the require clause are not equal to the values stated in the rule. This means that even if these fields were previously validated individually by some other rules, this rule may still invalidate the message. E.g., if the value of the field *morfologi* in the cancer message that is being validated equals 1530/39, or *messageType* is not ‘K’ or ‘R’, the values on the right-hand side of the require clause are immaterial and the cancer message is valid. If it equals 1527/39, however, and the value of the field *messageType* equals ‘K’ or ‘R’, then the right-hand side conditions must be applied because the left-hand side requires it. In this case, if the value of the field *topografi* does not start with ‘44’, the cancer message would be invalid. The rule would then output an error message describing the cause of invalidation.

After all message validation rules have been applied to the message, if none of them were violated, the message is valid. The message will then go on to be aggregated.

Cancer Message Aggregation

Before the messages are aggregated, all messages needs to be allocated to a cancer case. This is currently done manually by medical coders at the CRN. Note that a patient might have several cancer cases, and the relation between a cancer message and cancer case might not be straight forward. Thereafter, if the message is valid according to all applied MVR*s*, the fields of the message will be integrated into the corresponding cancer case by applying *Message Aggregation Rules* (MARs). For each of the fields in CancerMessage an MAR exists. The

MARs choose which types of messages are to be included, and apply calculations to the values of the corresponding CancerCase of a given CancerMessage, the result of which sets the value of the field in question in the CancerCase. One example of an MAR is for the field *diagnoseDato* (diagnosisDate) that states that the value of this field in a given CancerCase should be *the earliest of all the event dates* found in any of the related CancerMessages. By "related", we mean the CancerMessages that carry the same value in the field *cancerCaseNumbers* as the CancerCase in question. These are messages that are related to the same occurrence of cancer for the same patient. After a patient has visited more than one medical source for treatment or follow-up of one particular cancer, there will be just as many messages submitted to the CRN regarding this cancer case, and they may have more than one distinct value of the diagnosis date field. These messages must be aggregated[3], and there must be some decision made as to which value to compute for the resulting CancerCase—this is the function of the MARs.

Cancer Case Validation

The last step in the cancer case registering process is yet another validation for the aggregated cancer cases. Like with cancer messages, there might be errors in single fields of a cancer case, and there can be errors in consistency. I.e., one field that may by itself have a valid value may not be valid when viewed in context with some other field. This can happen because of the nature of messages; they are gathered from different sources that each fill in their respective fields. Thus, when all of these messages come together in aggregation, inconsistencies must be expected.

The Cancer Coding Rule Model

As described above, the CancerCodingRule class has one operation, which can be regarded as a Boolean (true or false) function in the case of validation rules and a more generic function in the case of message aggregation rules. In addition to the rule operation, CancerCodingRule (regardless of type) has several important attributes. First, they refer to a set of fields that are constrained by its function. Second, each rule is associated with a set of four dates: (1) *activationStartDate*; (2) *activationEndDate*; (3) *diagnosisStartDate*; and (4)

[3] A note on the term "message aggregation": it is being considered by the CRN to be changed to "case calculation". For one, the word "message" makes it unclear that it is in fact the cancer cases that are the results of the aggregation. Second, the "aggregation" term does not express that the process involves calculations of the cancer case field values.

diagnosisEndDate. Dates (1) and (2) tell when the rule begins and ends its service, respectively. This means that a rule that is past its activationEndDate must never again be applied to any cancer message or case, and one that has not yet reached its activationStartDate cannot yet be applied. Dates (3) and (4) refer to the diagnosisDate of CancerMessage and CancerCase (it is a CommonField). These dates give the time period in which the rule can be applied; any cancer data objects that have a diagnosis date that falls within this period would be subject to the constraints of this rule. E.g., a cancer message with a recorded diagnosis date of April fifth 2017 would be checked by an MVR that has a diagnosisStartDate of June tenth 2015 and a diagnosisEndDate of April sixth 2017, but not by any rule that has a diagnosisEndDate that is earlier than April fifth 2017.

In summary, Figure 2-4 from the domain analysis by Wang *et al*. illustrates the entire CRN domain as a UML class diagram [12]:

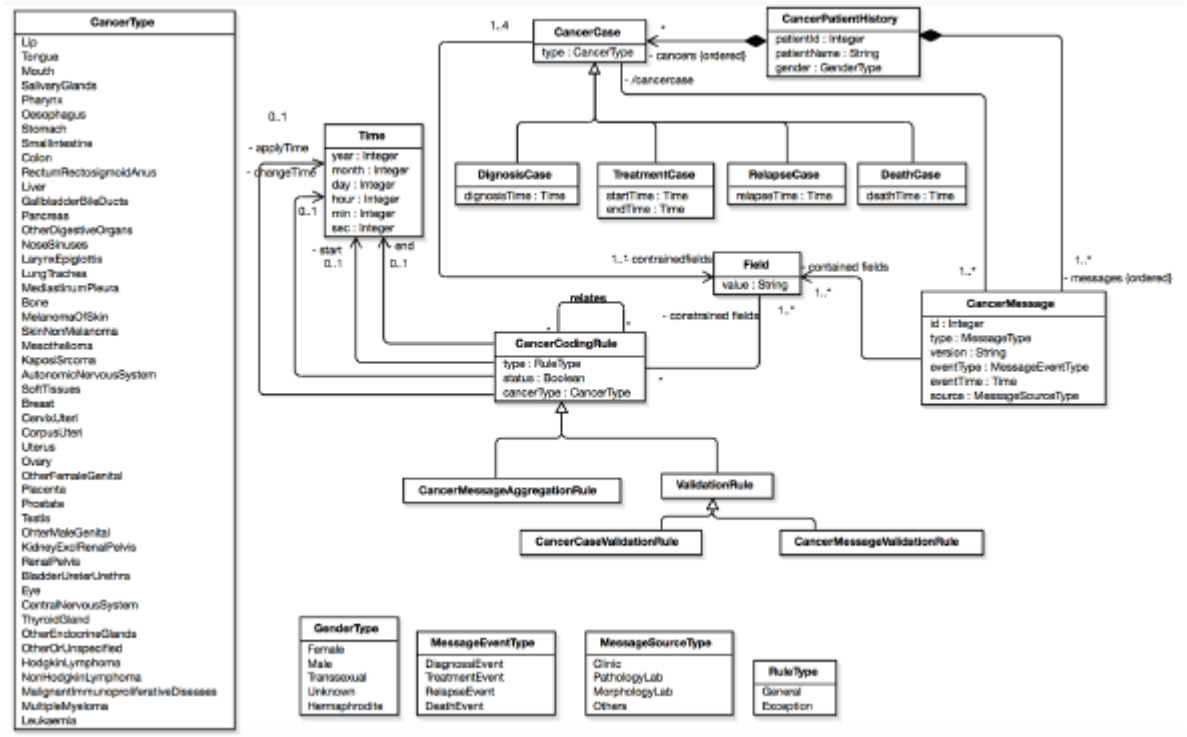

Figure 2-4: Domain Model of the CRN (simplified).

Note that this model does not display all of the fields that actually exist in each of the classes; e.g., recall that *CancerMessage* class has 64 fields. Rather than providing a complete description of the domain, this diagram is meant to give an overview of the concepts and their relations to one another.

2.2.5 Challenges of the CRN

In the domain analysis of the CRN, the researchers identified five key research challenges rooted in the implementation of the current system. They can be summarized as follows: (1) *low level of abstraction in the system*; (2) *lack of formal coding rule definitions and central rule repository*; (3) *rule application in the validation and aggregation processes are exhaustive and thus inefficient*; (4) *the four time dimensions (i.e., diagnosis, treatment, relapse, and death) are identical in the messages and their associated cases*; and (5) *relations between rules are not explicitly captured*. This section explains challenges 1, 2, 3, and 5, and how they are to be solved by the MBF4CR. This is worthwhile because the proposed CIA approach may need to operate within the constraints of the MBF4CR, which could dictate its implementation, and because challenge 5 is the very challenge we aim to address.

Challenges 1 – 3

To elaborate on challenge 1, a low level of abstraction in the system refers to the way coding rules are defined: they exist partly as source code, partly as database triggers and tables, and partly as written instructions to *medical coders*. The medical coder, a central role in the CRN, is an associate responsible for applying rules to messages and cases for validation and aggregation. This role is currently separate to that of the *medical programmer,* who is the one who implements the rules, in either the application layer or the database. One major issue with this is that when a new medical coder enters the CRN, it is challenging from them to acquire domain knowledge. They are required to study the concepts of cancer messages, cancer cases or cancer coding rules in order to understand the aggregation process.

Challenge 2 is related to the scattering of rules across different means of representation, i.e., source code, databases, and lookup tables. As rules are hard-coded into the system (i.e., their values and conditions are written as constants in the code), when it is changed or deleted, or when some new rule is added, much rework in the form of regression testing in addition to the testing of the new or modified rule must be done.

Challenge 3 stems from the fact that the system represents *CancerMessage* as a single class for all the different sources of messages. As previously mentioned, this means that many fields in a given message may be empty. The system has no way of intelligently selecting which rules should applied based on what fields might need checking, but requires that *all* are

applied in sequence. It should be noted that the medical coders' task of rule application is not automated.

2.2.6 Tool Support Provided by MBF4CR

In addressing challenges 1 through 3, the project has thus far has produced a web-based tool named Cancer Registry Validation and Aggregation Tool (CRVaT) developed by Simula in collaboration with the CRN [13] CRVaT has five main functionalities: (1) the *Extraction Tool*, which performs automatic generation of UML diagrams from database schema; (2) the *Tagging Tool*, which associates *tags* with rules; (3) the *Validation Tool*, which selects and executes relevant message validation rules; (4) the *Aggregation Tool*, which selects and executes aggregation rules on messages before integrating them into a case; and (5) the *Transformation Tool*, which updates the CRN database following aggregation. In addition to these the framework employs a small set of external tools.

The heart of the tools' functionality is the UML models of the CRN's system. These models, parts of which have been presented by figures 1 – 3, are initially converted automatically from the database schema (of cancer messages, patients, and cases) using the extraction tool. They are then refined manually. In non-model-based software development projects, UML models tend to be mere disposable artifacts of the requirements specifications. They are intended to describe the system on increasingly lower levels of abstraction until it has been implemented, after which they are rarely used. In the MBF4CR however, the models are first-class citizens; they are taken as input by the automatic validation and aggregation tools. The UML class diagrams alone are not sufficient as input for the tools to do their jobs, however; additionally, the framework relies on the *Object Constraint Language* (OCL).

Object Constraint Language (OCL)

OCL, like UML, is an Object Management Group (OMG) standard. It is a declarative language for specifying constraints on UML models or other *Meta Object Family* (MOF) metamodels in a formal manner. Together with OCL, UML can facilitate specification of systems/software in a more precise way. OCL constraints have been widely applied together with UML models to enable model-based testing and consistency checking of UML models. In the MBF4CR, it is instrumental to the solution to the problems surrounding rule definitions.

For the uninitiated, OCL is a formal language for writing expressions that specify invariant conditions that must be true for the system being modeled or for queries over objects described in a model [18]. In other words, using OCL, rules can be written for constraining the models of a system, e.g., defining valid values for fields or valid relations between objects given values of their fields. This is what the CRN's vast cancer coding rule set already does, so what is interesting about OCL? For one, UML class diagrams cannot fully express the constraints imposed upon the objects by the coding rules. I.e., the class models do not tell the full story about the semantics of each of their fields. With the UML-compliant OCL, code for constraining the fields can be generated automatically with tool support. Thus, there would be no longer need for medical coders to understand implementation details—they would only have to know OCL. This is one reason why work has begun to rewrite the cancer coding rule definitions in the standardized and formal OCL. Furthermore, defining the rules in this uniform way facilitates a centralized rule repository that is independent of the implementation of the ACRS. To present an example of how a cancer coding rule would look in OCL, consider the validation rule example from 2.2.4:

morfologi = 1527/39 and messageType = K, R require topografi = 44.x,

would be written in OCL as

morfologi = '1527/39' and messageType in['K', 'R'] implies topografi startswith('44').

While the OCL rule does not look much different from the original one, it is now written in a strict syntax that can readily be parsed by supporting tools. The issue with the syntax of the original rule is that it is not consistently enforced throughout the rulebook of the CRN, and even if it were, it would require a custom parsing tool for code generation.

The solution to challenge 2 involves automating the process of selecting and executing rules for validation and aggregation. An automated solution based on the current implementation would be hard to imagine as sustainable. In the MBF4CR however, the class models and OCL provide a way. Written as OCL expressions, the rules can be evaluated by tools such as Dresden OCL [19], which the validation and aggregation tools are built upon. With the tagging tool, an OCL rule can be augmented with one or more *tags.* Tags are simply the names of the fields in the *CancerMessage* or *CancerCase* with which the rule is associated, e.g., if the rule specifies some constraint on the values of the fields *messageType* and *surgery*, the rule will be tagged with these. The validation and aggregation tools work by first taking as

input the UML class diagrams and the OCL expressions. Then, by looking at the tags of the OCLs, they select only the relevant rules required for a given message or case. Lastly, they validate the message or case according to the selected rules. In cases of invalid messages or cases, the tools create reports of which rules were violated.

Challenge 5 – Change Impact Analysis

Last, but certainly not least, challenge 5 is the challenge of predicting the consequences of making some *change* in the CRN domain. This is the challenge that the CIA approach proposed in this thesis aims to tackle. As part of the domain study a change impact analysis[4] was conducted, uncovering three main classes of changes that can occur in the CRN: (1) *Data Change;* (2) *Domain Knowledge Change;* and (3) *Cancer Coding Rule Change* [12]. In order to elaborate on this challenge, it is necessary to look into the various changes that can occur at the CRN, and how they are experienced by their associates. The following section reviews the impact analysis study of the MBE4CR.

2.2.7 Change Classification in the CRN

Data Change

A data change is the change of cancer messages that occur as the sources provide updates, meaning that the *values* of the fields of the message can change. Naturally, as the aggregation process integrates fields from the message into the cases, the data in a cancer case will be affected along with the data in a relevant message. Updated messages that are received must be regarded in the same way as new messages, and thus a new aggregation must be performed as well. As usual, the validation of messages, followed by validation of the aggregation, is required afterwards. A change to any cancer message will therefore ripple to its related cancer case, as the subsequent aggregation may also cause the case to change.

Domain Knowledge Changes

Domain knowledge changes are changes relevant to cancer messages and cancer cases. I.e., these are changes to the system's representations of messages and cases, *not data values*. In

[4] Note that here the term "change impact analysis" is used in a different sense than in the rest of this thesis. It here refers to the *study* of the changes to the domain and the impacts of such changes to business processes—*not* to an automated CIA approach.

other words, they are changes to the domain model. New domain knowledge due to, for instance, new research findings can give reasons to add, remove, or modify fields for cancer messages and cancer cases. Such changes require the chief medical officers to check if it is necessary to make changes to the coding rules. For modified fields, rules that constrain the data in these fields must be reconsidered; for added or removed fields, rules may have to be added or removed as well.

Cancer Coding Rule Changes

Cancer coding rules are changed in terms of modifying some cancer coding rule, e.g., by removing, adding, or replacing one of the fields that it constrains, or changing the required values of any of the fields constrained by a particular cancer coding rule. Regardless of which part of the rule is changed, the original rule would continue to exist in its original definition, but would be *deactivated* from a certain point in time at which the new definition of the rule would be *activated*. Deactivation simply means that the rule must no longer be applied. This is the most basic impact of a cancer coding rule change. Its impact does not stop there, however—a host of cancer messages and cancer cases must now be re-validated or re-aggregated (depending on the type of cancer coding rule that was changed), as the values of the fields that the old version constrained may now be invalid according to the new rule definition.

Implicit to the seemingly straightforward rule system is the notion of *relations* between rules. As part of rule changes, these relations may also change. No formal classification or definition of these relations currently exists, but one type of relation that is readily apparent is *order*. As a hypothetical example, recall the first example of the rule that checks the age value, and assume that it is the very first rule that the system applies to the message. It is itself not dependent on any other rules, but every single other rule is at least implicitly dependent on it passing its check (otherwise they will not be applied). Some other rule that is applied later in the sequence may explicitly refer to values of fields that have already been validated by previous rules, but may still fail to pass its check. In other words, applying some rules *require* applying other corresponding rules first. Thus, there may be interactions between rules that cause changes to ripple further than just the replaced rule and the cancer messages or cases that were constrained by the rule. Conceivably, a rule change could recursively apply to any number of other rules and the data objects constrained by them.

2.2.8 The Motivation for Automated CIA in the CRN

In light of the knowledge gained from the study of the CRN domain, we here elaborate further on the outline on the problem we gave in section 1.1, and summarize the motivation for our proposed CIA approach.

To reiterate the general motivation for automated software CIA: it is of importance to the process of maintaining and evolving software because changes can readily ripple across the entire system and individual developers struggle with predicting the entire set of side effects. Indeed, experiments have shown that developers are inaccurate in their judgments on how changes would affect the system as a whole [3, 20]. While these findings may not generalize to the CRN and its medical coders and medical programmers, it does cause concern. The current situation with CIA in the CRN is analogous to the manual software CIA process described in section 2.1.1: the responsible associates have to painstakingly analyze change impacts in a manual fashion as tool support does not exist. The main difference is that changes will not directly happen to the software; the relevant changes in the CRN are done to persistent cancer data (cancer messages and cancer cases), to its domain model, and to cancer coding rules. Predicting the impacts of the latter is particularly complicated, as it ripples to the other rules as well as the persistent data. The CRN associates cannot be expected to predict every possible impact and consequence, which is why thorough testing following rule changes must be done. While the need for this testing probably cannot be eliminated, it could certainly be done more efficiently if it were guided by automatic predictions of change consequences. A benefit could be narrowing down target data for examination, and target cancer coding rules for re-validation and re-aggregation. This could reduce the time and effort required to handle side-effects considerably. Ultimately, the purpose of providing automated CIA tool support in the CRN is to support the data quality measures applied therein. This is because the greatest conceivable risk of any change is that invalid cancer data may secretly exist in their databases, compromising the validity of the research they do. While such errors rarely occur, and are always discovered eventually, they can cause the need for rolling back the errant database. Needless to say, rollbacks cause the loss of much time. To be effective, the tool must clearly support the analysis of all classes of changes, which appear to us as entirely unique in the context of software CIA (the word “software” even seems out of place). To our knowledge, no CIA approach for a cancer registry (or any kind of medical registry) application domain has been proposed. Furthermore, due to the specificity of its domain, the

CRN has ruled out the use of any existing commercial off-the-shelf CIA tool, and instead chosen to investigate a custom approach in a research setting alongside the MBE4CR project.

2.3 Related Works

We applied a set of criteria judging the applicability of a given CIA approach to the CRN CIA problem when reviewing several existing software CIA works. The criteria are: (1) perceived similarity of context or domain of the approach application to the CRN domain; (2) perceived similarities of approach implementation with the MB4FCR software platform; and (3) where applicable, performance measurements or theoretical complexity in terms of EIS computation. The criteria narrowed down the reviewed works to four works that provided inspiration for our proposed CIA approach.

2.3.1 Rule-based Impact Analysis for Heterogeneous Software Artifacts (Lehnert et al.)

This approach proposed by Lehnert et al. is (according to Lehnert's taxonomy) a combined-scope approach. It incorporates aspects of architectural model-based, requirements model-based, and explicit rule-based techniques. The work addresses the issue of change impact across different types of software artifacts, i.e., source code, unit tests, and system models [8]. The two major challenges in analyzing change propagation between these artifacts are: (1) interconnecting the different types of artifacts and (2) propagating changes across them.

The approach unifies and stores all artifacts of the system set. After the artifacts are unified, rules for traceability discovery are applied to record *dependency relations* between them as *traceability links*. The proposed impact analysis algorithm is an improved version of dependency analysis; the authors note that "pure" dependency analysis is too imprecise and creates too many false positives. The hypothesis underlying their impact propagation technique is that "the interplay of change type, dependency type, and the types of involved artifacts determines if and how a change ripples to related artifacts". In summary, the proposed approach consists of four main steps:

1. Transformation of artifacts into unified Eclipse Modeling Framework (EMF)-based models and import into model repository.

2. Rule-based dependency analysis.

3. Classification of change type.

4. Rule-based propagation of impact propagation.

A feature of the approach claimed to be unique is that it does not "cut off" the change propagation according to some predetermined distance, unlike dependency graph-based techniques commonly used in source code-based CIA. Another novel feature is that the approach provides developers with information on how the artifacts should be changed. The final point of interest is that Lehnert et al. boast significant performance advantages in terms of impact set sizes (up to 2100% reduction), and precision (=>80% versus <20% on average) over dependency analysis techniques that employ distance measures to limit change propagation. This suggests that their rule-based impact propagation algorithm is efficient.

Employing the *exact* same design and implementation in the MBF4CR as Lehnert et al. was not considered feasible. This is because the approach does not appear to be able to consider impacts of any kind of change to individual persistent data objects, only the impacts of software artifacts expressed in source code or models. However, the overall structure of the approach along with the algorithm and dependency and impact propagation rule concepts appeared adaptable to the CRN domain due to their generic nature. Furthermore, the approach bears some conceptual similarity with the MBF4CR platform with its model-based features. The equivalent to step one in the approach of Lehnert et al., the transformation of artifacts into unified EMF-based models, is inherently being done by the MBF4CR. The second step, dependency analysis, appears at least partially supported by the MBF4CR; relations between cancer messages, cancer patients, and their corresponding cancer cases are already captured explicitly in their UML models (yet implicit relations exist and are not captured).

2.3.2 A Taxonomy of Change Types and its Application in Software Evolution (Lehnert et al.)

This article is a work related to "Rule-based Impact Analysis for Heterogeneous Software Artifacts" presented in 2.3.1. It does not propose a CIA approach, but addresses the lack of consistent classifications of types of change operations in the software CIA literature, and proposes a generic, graph-based change taxonomy for CIA [21]. The authors lament that

existing works typically limit their change classifications to specific aspects of software (e.g., source code or architecture). The change taxonomy is applied in the related CIA approach of Lehnert et al., which is why it is included in our review.

In this taxonomy, the change types are classified as either *atomic* (*add, delete, update*) or *composite* (*move, replace, split, merge, swap*). Composite changes may consist of sequences of atomic and composite operations. The approach considers the system on which CIA is performed as a labeled, directed and (possibly) circular graph, in which arbitrary software artifacts (diagrams, classes, etc.) represent the nodes. Dependencies between these artifacts are represented by the edges. The attributes of the artifacts are property labels in the graph nodes and edges. Any changes to the system are implemented as generic graph operations. E.g., the composite change type *move* is the equivalent of moving a sub-graph to another node. If the moved node was a class, all its attributes and methods represented by their own nodes connected to the class node would follow.

This work provided a reason to apply our own change taxonomy in our proposed CIA approach: it would be required in order to implement an approach similar to that of Lehnert et al. In our case, the change model was already defined based on the study of the CRN domain, but it was refined to be clearly separated into atomic and composite change types—this resulted in a simpler model. We present our change model in chapter 3.

2.3.3 Assessing Impacts of Changes to Business Rules through Data Exploration (Embury et al)

This work addresses CIA techniques on business rules [9]. Particularly, the change to a business rule can have unpredictable consequences to persistent data, or to other rules. The evolution of business rules is a costly exercise, due to the sheer number of rules that a given organization may have and that they often have to be implemented by several different systems. While tools for predicting impacts to software components and for identifying conflicts between rules exist, *data level* impact techniques have not been studied. The authors define data level impact as “where new rules force a reinterpretation of the data that describes the current and past state of the organization”.

The approach models changes in as occurring in three forms: (1) the addition of a new business rule to the system; (2) the deletion of a rule from the system; and (3) the amendment

of some aspect of an existing rule. “Impact” in this context is consequences of a change to a business rule to other rules (i.e., which rules must be changed to accommodate the new/updated rule), and the persistent data that are constrained by the affected rule(s) (i.e., which records must be cleaned or deleted in order to be compatible with the change).

The approach of Embury et al. is interesting as it claims to be the first to analyze the impacts of changes to business rules, specifically *constraint business rules* (CBRs), to persistent data objects constrained by such rules. Constraint Business Rules restricts the state of the organization, as represented by the data (e.g., customers who have failed to meet the deadline for payment more than twice are not eligible for membership of a loyalty program). Changes to CBRs impact the system by invalidating previously valid data values, while other previously invalid values may become acceptable. These impacts may sometimes be intended, but the negative case is that values that are acceptable in the real-world change to unacceptable in the system. In the inverse case, values that are unacceptable in the real-world become acceptable in the system. When considering cancer coding rules—particularly VARs (see section 2.2.4)—in light of this problem area, they can be regarded as a special case of CBRs.

The approach implements impact analysis as queries on hypothetical relations that select values that were invalidated or validated by a change to a CBR. The rules for impact analysis are the queries themselves, implemented in structure querying language (SQL). While Embury et al. do not provide any performance measurements of their approach, the queries appear to produce highly precise sets (i.e., the sets should rarely include objects that should *not* be impacted by the given change). Also, it could enable the computation of an estimated impact set without the need for first producing a starting impact set. Hence, we propose a query-based technique for selecting impacted objects in section 3.4.1. We measure the performance of our implementation in terms of precision and other metrics in chapter 4.

2.3.4 Impact Analysis and Change management of UML Models (Briand et al.)

The approach proposed in this work addresses the general issue of changes to UML models, which cause the need for subsequent changes in the same or related models. Specifically, affected diagrams should be automatically identified to (1) keep them updated and consistent;

and (2) assess the potential impact of changes in the system. The approach presented focuses on the CIA of UML *Analysis* or *Design* models.

The authors note that this approach also addresses a fundamental weakness of code-based CIA: the changes identified by such methods must be implemented before an IA can be performed. With a model-based approach, the impacts can be observed prior to any implementation, and allows decisions to be taken on which changes to implement. On the other hand, model-based approaches may provide less precise results than that of code-based approaches, as unexpected impacts may appear at implementation time (this goes beyond the scope of this paper).

The authors decompose the CIA of UML design models into the following problems:

1. Automatically detect and classify changes across different versions of UML model. Each model element change is classified according to a *change taxonomy* in order to associate IA rules with each type of change.

2. Verify the consistency of changed diagrams, i.e., find structural inconsistencies between diagrams (e.g., a class instance in a sequence diagram that is not present in the class diagram). Inconsistencies may be automatically modeled by and detected by set of *consistency rules.*

3. Perform automatic IA to determine potential side effects of changes. An *impacted element* is a UML model element whose properties or implementation may require modification as a result of changing another model element (i.e., its properties). *IA rules* determine what model elements could be directly or indirectly impacted by each model element change.

4. Prioritize the results of the IA according to the likelihood of occurrence of predicted impacted elements. Measures of distance between the changed elements and potentially impacted elements are defined to indicate which elements should be checked first. The greater the distance, the less likely is the element to be impacted.

Each impact analysis rule is specification of how to derive several collections of elements corresponding to elements of different types that are potentially impacted by a particular change. As stated by Bohner et al., A common assumption made in distance-based CIA (that

has yet to be empirically validated) is that “if direct impacts have a high potential for being true, then those farther away will be less likely” [22]. The distance between a changed element and a given impacted element is defined by Briand et al. to be the number of given IA rules that had to be invoked to identify this impacted element.

The results of the evaluation of this approach show that it produces linear growth of the EIS, rather than the expected exponential growth of connectivity graph-based approaches. Briand et al. do not report precision and recall measurements of the approach, but postulate that the linear complexity is by virtue of high precision provided by their IA rules. This is the main reason for including this approach including this approach in our related works. It lends further credence to the explicit rule-based and model-based software CIA approaches as being the most effective and efficient. Hence, we choose to proceed along this path with our proposed CIA approach for the CRN.

2.3.5 Summary of the Literature Review

We came to the conclusion that, to our knowledge, no CIA approach in existence addresses the issue of analyzing changes to persistent data that are under the constraint of the same kind of medical coding rules that is used by the CRN. However, the one approach found that addresses an issue closest resembling this also appears as highly efficient and conceptually simple. Furthermore, the generic combined-scope approaches appears amenable to adaptation to the CRN domain. In one way, we assess the problem of CIA in the CRN as simpler than that of CIA for heterogeneous software artifacts; we only have to deal with changes to three concepts in one domain model. Thus, we aim to design, implement and evaluate a custom CIA solution based on concepts found in the existing approaches studied, with adaptations to address the CRN-specific issues.

3 The Approach Framework for CIA in the CRN

This chapter presents the proposed CIA approach, and discusses crucial design choices made regarding the approach *framework*. The framework consists of three main concepts: (1) the CRN domain model that was given by the MBE4CR project and presented in chapter 2.2; (2) the *change model*, an explicit definition of changes that occur in the CRN domain based on the domain study; and (3) the *Change Impact Analysis Rules* (CIARules), a two-component rule framework for (a) defining *dependencies* between objects in the domain model and how the changes in the change model would propagate to each other, and (b) what the *impacts* of these changes would be. The approach employs an algorithm for computing estimated impact sets that operate by recursively applying the CIARules to CRN concept objects, starting with the object to change. In section 3.5 the definition of this algorithm is given along with a theoretical complexity analysis. The proposed approach can be summarized as a combined-scope approach, being both explicit rule-based and architectural model-based and using other select techniques from source code analysis.

Note that the focus of sections 3.1 through 3.5 is on the theoretical aspects of the approach—they do not present a concrete software implementation (but will discuss some considerations for any hypothetical implementations). The chapter is concluded with section 3.6 presenting the prototype software tool implemented for the purpose of evaluating the approach framework.

3.1 High-Level Design

This section discusses the high-level requirements of the approach, i.e., what functionality the approach must provide and any important qualities that are expected by the CRN. Also, questions that arise regarding the techniques that should be applied in the approach are discussed. Further sections in this chapter will elaborate on the approach concepts.

3.1.1 Requirements of the CIA Approach

The approach must support the impact analysis of any class of change that was uncovered by the domain study, i.e., data changes, domain knowledge changes, and cancer coding rule

changes. This includes any subtypes of changes. Thus, the approach must have explicit knowledge of the types of concept objects that can be changed, and types of change that could be proposed. These are the roles of the CRN domain model and the proposed change model in this framework, respectively. The framework must provide capabilities for an application to be built, with which CRN medical coders, medical programmers, and other CRN associates can analyze proposed changes *without actually changing any real objects*. The framework must provide *automated* CIA, meaning that its users should only have to manually input the change requests in order to perform analyses.

Performance

Any algorithms employed by the framework must scale within reasonable bounds (assuming an optimal implementation), meaning that they must terminate within a time period that is orders of magnitudes shorter than that which is normal in the current practice (e.g., it should finish in minutes rather than hours, or seconds rather than minutes). At the same time, the approach must not consume computing resources (i.e., CPU time and primary memory of the systems running the tool) in such quantities that its use would disrupt other business process at the CRN. The tool must not only be faster than the current practice; it must also provide analysis results (EIS) that are at least as precise in terms of actually impacted objects, and accurately describe impacts with regards to real-world consequences.

3.1.2 Change requests and Starting Impact Sets

What should the solution accept as input and what would it return given that input? The input, the change request, would in the context of the CRN come in the form of a data change, domain knowledge change, or a cancer coding rule change, with a corresponding value with which to replace the old. The value here would be dependent on the type of object to change, e.g., a data change would only consider values that are of the same data type of the field that is to be changed. The SIS (see section 2.1.1 for the introduction to impact sets) given by the CIA approach, would then contain: (1) some values of messages and cases, (2) fields of messages and cases, or (3) coding rules; or possibly (4) all of the above. Then, the CIA tool must determine what other elements depend on those in the starting impact set, producing the EIS.

3.1.3 Computing Estimated Impact Sets

As a basis for a function that compute estimated impact sets, one could imagine the CRN domain as a graph or matrix of cancer message, cancer case, patient, and cancer coding rule objects. Computing the starting impact set should then, in principle, be a search through this graph, finding objects sharing some relation to the object in the change request. Similarly, finding the related objects for the EIS would then involve gathering objects adjacent to the ones in the starting impact set. However, recall that the concept of relations between rules has yet to be formalized. Thus, in the supposed graph there would be no edges connecting the *CancerCodingRule* objects to other rules. This raises questions as to how to implement a rule relation: what attributes will the rules need to refer to other rules? Can it be done without introducing new attributes in the model? This question of how to implement relations or dependencies also applies to the other classes of CRN concepts. The proposed approach addresses the question by using only *existing* relations between objects that are found in the domain model, and does not need to alter the domain model. E.g., one of the most obvious but most important dependencies is derived from the relation between a cancer message and a cancer case object. This relation is defined by the equality of the value of the common field *cancerCaseNumber*. Thus, if one changes a cancer message (a data change), the framework expects that the cancer case with the same *cancerCaseNumber* could be impacted. In many cases, such as with cancer coding rules, there may also be relations that are less obvious, meaning that objects could be implicitly related and thus possibly affected by a change. The approach provides support to define also these relations (both hypothetical and known) without adding any features to the model. The method by which this is achieved is presented in section 3.4.1 about *dependency rules,* a subclass of the CIARules.

3.1.4 Algorithms and Data Structures

Implementing the solution as a graph or matrix may however not be desirable. As has been demonstrated by classic software CIA techniques using dependency graphs or call graphs, the functions computing impact sets by traversing the graphs must sacrifice precision and recall for the sake of controlling the exponential complexity of the search algorithms. Furthermore, just because some object is related to the changed object does not mean that the related object is *actually* impacted—this means that such techniques tend to be imprecise as well. Still, the graph is an intuitive and flexible data structuring technique, supporting a generic

representation of interdependent and possibly heterogeneous object networks. It has for this reason been employed by recent approaches that happily have found ways to reducing the algorithm complexity while sacrificing less precision and recall (e.g., see sections 2.3.1 and 2.3.4). The proposed solution to the data structures question is again the dependency rules, assisted by the second type of CIARule, the *impact rule*.

The impact analysis algorithm chosen for the approach is of custom design but, similar to the algorithm designed by Lehnert et al., applies dependency rules to construct a graph (in our case a *tree*) of interdependent objects. Our algorithm will be presented in 3.5.

3.2 The Role of the CRN Domain Model

The approach relies on the CRN domain model for knowledge about the concepts that can be changed. Specifically, the knowledge used includes the values of certain features of each changeable concept class (i.e., CancerMessage, CancerCase, and CancerCodingRule). Together with the change model this provides the CIARules, which will be introduced in section 3.4, the necessary data model for determining dependencies between concept objects and the impacts of each type of change. The CRN domain model is the interface object model of our approach. An implementation may decide to use a subset of the domain model as the internal object model; not necessarily all classes are required to perform CIA in the CRN.

As previously mentioned, the approach does not need to alter any class definitions (i.e., adding or changing features) in the domain model. The motivation behind this design decision was that the existing domain model should be able to be used by a potential software prototype deployed in the MBF4CR.

3.3 The Change Model

To assist the CRN Domain Model provided by the MBF4CR and presented in 2.2, this section introduces the *change model.* The primary purpose of this model is to provide an explicit definition of the types of changes identified in the CRN (i.e., data, domain knowledge, and cancer coding rule changes, including subtypes) and their relations to one another. This is used by the proposed CIA approach as a basis for the rule-based change propagation. Specifically, the model provides the CIARules with information about proposed changes, which they use for determining which objects should be impacted given the specific type of

change and type of CRN concept object. Therefore, the classes in the change model are responsible for storing information about changes in addition to their names (which by themselves are very useful), including the object that is to be changed, and the field or other attributes to be changed in a CRN object. Furthermore, the model specifies which change requests are valid for each type of change in order for the proposed changes to make sense (e.g., a new cancer coding rule would not be valid as a request for a new value in a cancer message). Clearly, the greatest concern of this model is to accurately represent the changes that occur in the real world. This makes the model also useful for a researcher who is interested in understanding the CRN domain (this was in truth the original motivation for its creation).

As a final note on the design of the change model, it should be emphasized that it does *not* specify the impacts of the changes; this responsibility has been left to the impact rules. The main reason for this is that the impact specifications were from the beginning expected to be changed frequently as the understanding of the CRN domain would be evolving over the course of this project. Impacts of changes, along with the relations between objects, were the greatest sources of uncertainty about the CRN domain. As a system model change tends to ripple to other models and existing implementations of the model (in case it is implemented manually), decoupling impact and dependency logic from the change model could potentially save time on implementing changes based on feedback on the approach later gathered from CRN stakeholders.

To enable automatic code generation of the change model in an implementation of the proposed CIA approach, the model was drawn using the Eclipse Modeling Framework (EMF) [23]. This provides the facilities required to perform code generation by using a supporting tool (e.g., Eclipse Neon [24]). Optionally, the model can be implemented by hand, i.e., using classes written in Java or any other object-oriented programming language, as long as the main change types and their attributes and operations are implemented. With manual implementation, the developer obviously chooses to forego benefit of model-driven code generation: that any changes to the model would not automatically be reflected in its implementation. This applies both ways; if the implementation deviates from the model, the model needs to be manually updated afterwards (unless it is acceptable that the model becomes obsolete).

While the model was actually drawn before conducting the literature review, motivation for its continued use in the design (and eventually in the prototype implementation) was reinforced by Lehnert et al. Recall that the authors stated that the type of change and type of changed object can be used to limit the change propagation more precisely than by using a simple cutoff value (see 2.3.1). They showed evidence that this hypothesis holds true by presenting favorable performance numbers. This hypothesis intuitively appears to apply also to changes in the CRN domain; from the domain model we know what objects relate to one another (e.g., cancer messages relate to a cancer case), and a change to any one of these objects must therefore be expected to ripple to a related object. The rippling should only happen given the right type of change for the changed object. For instance, only a data change would ripple from a cancer message to a cancer case. Furthermore, the data change can only begin with a cancer message, unlike the domain knowledge change which could begin with either class. Thus, this change model is core to the proposed CIA approach, as it gives the explicit types required to recognize what kind of changes can be applied given a certain type of CRN concept object. Naturally, the change model must coexist with the CRN domain model, as this defines the concept objects that can be changed.

Figure 3-1 shows the change model in its entirety. The following section will explain the main classes of the model.

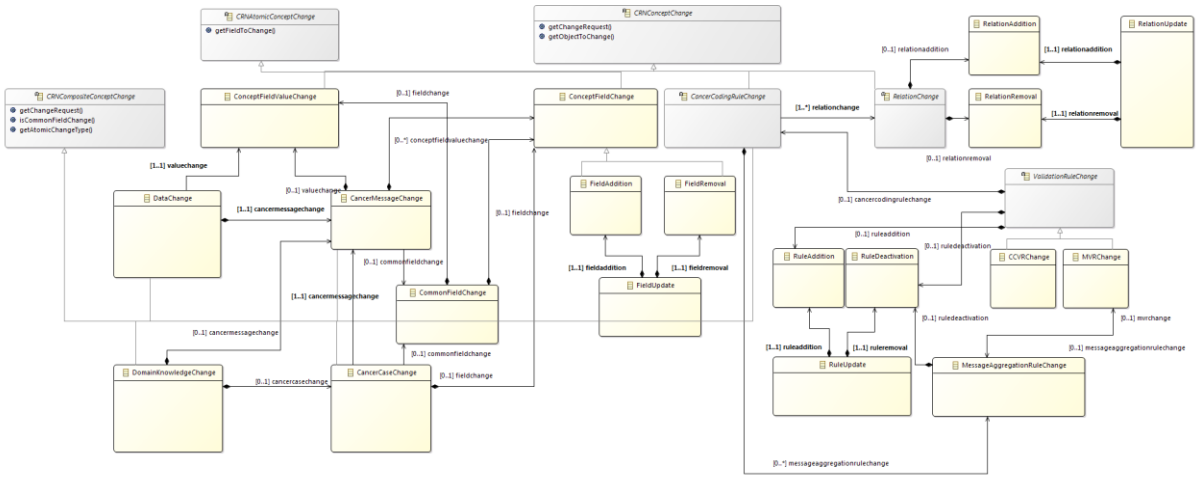

Figure 3-1: The Change Model.

3.3.1 The CRN Change Hierarchy

The change model is first and foremost a classification of the changes that are identified in the CRN domain. Thus, every class of change that is specifically related to any of the three main CRN concepts, cancer message, cancer case, and cancer coding rule, is a subclass of the top-level abstract class *CRNConceptChange.* On the next level of the hierarchy are the abstract classes *CRNCompositeConceptChange* and *CRNAtomicConceptChange*. As the name implies, CRNCompositeConceptChanges are change types that are composed of other changes; specifically they can be composed of CRNAtomicConceptChanges, and other CRNCompositeConceptChanges. CRNAtomicConceptChanges on the other hand, are *not* composite, meaning they represent only a single, atomic modification to a given CRN concept. Being abstract classes, these classes do not exist as actual objects, but define a template for how actual, concrete representations of these changes should look and behave.

3.3.2 CRN Composite Concept Changes

Inheriting from CRNCompositeConceptChanges are the three main classes of changes in the CRN domain. Recall from section 2.2.7 that these are Data Changes, Domain Knowledge Changes, and Cancer Coding Rule Changes. Each of them has a concrete class representation in the change model: *DataChanges*, *DomainKnowledgeChanges*, and *CancerCodingRuleChanges*. DataChanges and DomainKnowledgeChanges are further composed of the concrete classes *CancerMessageChanges*, and *CancerCaseChanges*, while *CancerCodingRuleChanges* are composed only of atomic changes. This section describes and explains the composite changes included in the model, along with the related atomic change types for each of them.

DataChange

The DataChange class can be composed of only a CancerMessageChange because a data change in the CRN domain occurs only as a change to some existing cancer message object in the database. Furthermore, recall that such a change would only happen to the *value* of a field in this cancer message. Thus, a CancerMessageChange is composed only of one object of the CRNAtomicConceptChange class *ConceptFieldValueChange.* ConceptFieldValueChange stores information about the DataChange: the field that is to be changed and the proposed new value of this field (the change request). The cancer message object to be changed is

referenced by the CancerMessageChange. Clearly, the types of change request that are allowed for the DataChange class must be any type that is represented in the attributes of the cancer message class and the common field class (e.g., integer, string, MessageType etc.).

The CancerMessageChange composing the DataChange can be related to a CancerCaseChange. This happens in the case where the FieldValueChange required a re-aggregation that resulted in a FieldValueChange of the related cancer case object. Recall that this is the only way in which a cancer case can have its attribute values altered.

The operations of all classes composing the DataChange are to simply provide access to the objects related to the change; the interface of the DataChange class (inherited from CRNCompositeConceptChange) gives the operations *getObjectToBechanged(), getFieldToChange(),* and *getChangeRequest()* for this purpose. Other useful operations include operations for determining the subtype of atomic change, and for determining whether the change is to a common field. The latter is important as the impact can vary considerably depending on whether or not the change is to a common field; this will be further elaborated on in section 3.4.2 about impact rules. Lastly, operations for retrieving the composing composite changes and atomic changes are provided.

DomainKnowledgeChange

Recall from section 2.2.7 that domain knowledge changes in the CRN can be regarded as changes to the domain model, specifically to the cancer message and cancer case classes. This means changes to the *fields* of these classes themselves, and not just to their values. A DomainKnowledgeChange object is naturally composed of either a CancerCaseChange or a CancerMessageChange; in either case, the composing class object would be further composed by the CRNAtomicConceptChange subclass *ConceptFieldChange*, and never a ConceptFieldValueChange.

The ConceptFieldChange represents any changes to the field other than a simple value change. Atomic changes include the *addition* or *removal* of fields (named *FieldAddition* and *FieldRemoval*), or an *update* to a field (*FieldUpdate*), which is modeled as a composition of a FieldRemoval and a FieldAddition (the old field would be removed before adding a new one with the changed attributes). In any case involving a FieldAddition, the accepted change

request would be a new attribute—a field. FieldRemoval operations would not expect a change request (the request would be the field to be removed itself).

The attributes of a field that are expected to change are the type of the field (e.g., change from a numeric type such as integer to a composite type such as string), and the name of the field. It is left to the specific implementation of the model the decision on whether both attributes can be changed simultaneously or only one at a time. In the real world CRN, changing either would be extremely unlikely however; domain knowledge changes usually (but still only on a yearly basis) come in the form of a field addition.

Operations provided by the DomainKnowledgeChange are the same as for DataChanges and all CRNCompositeConceptChanges; the differences would lie only in the return types of the operations. E.g., getChangeRequest() would return the new field that is proposed, and not some primitive type as in DataChanges.

CancerCodingRuleChanges

The CancerCodingRuleChange class is an abstract class composed of an atomic change of either the type *ValidationRuleChange* or *MessageAggregationRuleChange*, of course depending on the type of the rule that is proposed to be changed. Recall that concrete validation rules are represented in the CRN domain model by either one of two subclasses: MVR and CCVR. Thus, ValidationRuleChange is abstract and represented by the concrete classes *CCVRChange* and *MVRChange.* As usual, the type of the actual change instance would depend on the type of object to be changed.

Atomic changes that can occur to cancer coding rules are (1) changes to the fields that they constrain (i.e., the rule could be changed to constrain a new field, stop constraining some field, or constrain a different field instead of an existing one); and (2) changes to the constraints themselves (i.e., *valid values* for a given field would be changed). While these atomic changes could be classified distinctly, it is the practice of the CRN to replace old rules with new ones—the original definition of the changed rule would have to be preserved because it could be required to be in service for some time still. After this time the original rule would be deactivated. Furthermore, old rules must be kept in order for the CRN to be able to recreate the rule set from any given time period in history, so as to understand the changes of the rules, and to be able to recreate cancer data from this time period. The change

model therefore instead regards a rule change (RuleUpdate) as the composition of an addition of an entirely new rule (*RuleAddition*) and the deactivation of the old one (*RuleDeactivation*). This makes the model design simpler (i.e., it has fewer classes and thus fewer relations) than it would be if atomic changes were modeled explicitly. The reason why this simpler design would work in a real implementation is that the most basic impact of a rule update would be the same in any case: any persistent cancer data objects that could be constrained by the new rule could be invalidated or could need re-aggregation. Thus, the new rule would have to be applied on each of these objects to re-verify the constrained objects anyway. The effects of the *deactivation of the old rule* are a different matter, as the dependencies would in this case involve objects that would no longer be related to this rule. In that case, however, there would be no need to apply the old rule to analyze the impact. In-depth discussion on the known impacts of all change types is provided in 3.4.2.

As a side-effect of changing a cancer coding rule, its relations to other rules may be changed, giving rise to the change class *RelationChange*[5]. A CancerCodingRuleChange would be related to such a RelationChange if one of the following conditions occurs: (1) the changed rule ceases to be related to some rule to which it previously was related (*RelationRemoval*); (2) the rule becomes related to some rule to which it was previously not related (*RelationAddition*); and (3) one relation is replaced by another (*RelationUpdate*, modeled as a composition of RelationAddition and RelationRemoval).

This section has presented the change model, an explicit definition and classification of the changes known to occur in the CRN domain. The next section will demonstrate the use of this model along with the CRN domain model in the proposed CIA approach framework.

3.4 Change Impact Analysis Rules

The change impact analysis rules (CIARules) of the approach are that which defines and computes the interdependencies between objects of the CRN concepts, and analyzes the impacts of changes applied to them. The interdependencies (henceforth referred to as simply *dependencies*) of CRN concept objects are defined by *dependency rules*, while impacts of

[5] A note on the naming of this class: it is not prefixed by "Rule" even though it currently only relates to cancer coding rule changes. This is because it is conceivable that other change types could cause changes in relations between other CRN class objects. If this were to be discovered in the future to be true in the real world, the model would be extended with relations between other change types and RelationChange.

changes are defined by *impact rules.* This section will for both types of CIARule provide definitions and explain their role in the approach framework. All currently defined CIARules will be presented in section 3.4, with select rules used as examples. The CIARules form what Bohner and Arnold would refer to as the impact model of our approach (see section 2.1.2).

3.4.1 Dependency Rules

The dependency rule is a function that accepts as input a CRN concept object and a CRN change, and evaluates a sequence of *predicates* in order to determine which other objects would depend on the given object and thus be affected by the given change. The output of the function is the set of dependent CRN concept objects. As the dependency rules are applied recursively by the impact propagation algorithm, each object in the returned set may have another set of objects depending on itself (the CIA algorithm given in section 3.5 will elaborate on this).

It is important to now make clear the meaning of the term *dependency* within our approach framework. This term has in the CIA literature frequently been used interchangeably with other terms such as *relation* and *trace.* In our case, we choose the word *dependency* because it implies that objects in the CRN can *depend* on each other—they do not only *relate*. I.e., the state of any given object may depend on the state of another object that is related to it. E.g., the values of the fields in a cancer case would depend on the values of the fields of its related cancer messages (but not the opposite way—the cancer messages do not depend on the cancer case). This is what the dependency rules aim to capture because we are interested in retrieving *only* objects whose state may be affected by the state of the changed object. The rest would be unaffected even if they sometimes can be said to be related, and we do not want to waste time on them.

Functions of the Dependency Rules

Predicates are functions that specify one or more logical conditions on one or more values, and return either true or false depending on whether or not the values meet the conditions. The values evaluated by the predicates of a dependency rule are the types of CRN concept objects and changes (including change subtypes), and the values of the fields of the CRN objects. The conditions can be arbitrary, but include checking that the type of the change is valid according to the change model, or that the concept objects has a value of a certain field

within some range. Every dependency rule must begin by evaluating at least one predicate with the following two conditions: (1) whether or not the given change is any one of data change, domain knowledge change, or cancer coding rule change; and (2) that the concept object type is relevant for the type of change (even if the object is not the one that was changed directly). Then, if and only if both of these conditions are true, more predicates may be evaluated that must also be true in order for the rule to produce the set of dependent objects. The objects in this set will be subject to constraints defined by the dependency rule, defined as yet another set of predicates. If any predicate evaluates to false, an empty set is produced by the dependency rule. Examples of dependency rules will be presented in section 3.4.3.

The intent is that any software implementation of the approach framework defines these rules as queries in an appropriate language that retrieve the sets of objects potentially impacted by a change from the CRN databases. This particular design was inspired by Embury et al., who defined

3.4.2 Impact Rules

While dependency rules define *which* objects could be impacted by a change to some object, the impact rules define *how* those objects should be impacted. Impact rules are functions that, like dependency rules, accept as input a CRN concept object and change. They return an *impact,* a tuple of a change, impacted object, and a string (sequence of characters) that describe the impact. The change can be either the same change that was analyzed for the impacted object, or some new change in case the impact implies that the change is now of a new type. The impacted object would be an object previously returned by a dependency rule. The impact description is free text, possibly partially or entirely provided by some other functions that contribute to the impact analysis.

Functions of Impact Rules

Implicitly, the impact rule must have evaluated the same initial predicate (the one that checks the concept and change type) as the dependency rule that returned the set containing the impacted object. An impact would of course be computed if and only if this predicate evaluates to true (other conditions may apply as well).

Recall from section 2.2.7 that in the CRN, one crucial type of impact is that cancer messages and cases must be re-validated and re-aggregated following data changes and cancer coding rule changes. Thus, the impact rules must be responsible for applying any relevant cancer coding rules when given one of those change types. This is so that it can tell whether or not the change has caused any objects to become invalid (or valid) according to the selected cancer coding rules, and display changes to cancer cases caused by the aggregation rule functions. For each applied cancer coding rule there must be a statement of its result appended to the impact description, whether it was an invalidation, validation, or change of value of a field in the impacted object.

Impacts appear somewhat arbitrary in their definition because there can be many ways of expressing them in natural language. To keep impacts unambiguous, focus is on providing descriptions that state in plain terms "impact *i* happened to object *a* because change *c* implies event *e".* Examples of impact rules will be given in section 3.4.2, and will illustrate the various known impacts of changes in the CRN domain.

3.4.3 The Current Set of CIARules

By "current" it is meant that the CIARules presented here are those that, at the time of writing, have been defined and also verified and validated using the prototype software tool. One table is given for each main class of change, with explanations and discussion of select rules.

CIARule Tables

The tables presented below contain the complete set of CIARules in natural language. For dependency rules, each table cell gives (1) the subtype(s) of the change that is expected by the CIARule, (2) the expected type of changed CRN concept object, and (3) a description of how the changed object relates to other objects and which objects this includes. Where possible, the size of the returned set of the rule is estimated. The cell for each impact rule gives the same expected types as its corresponding dependency rule; its description gives any action that must be done for each of the affected objects, including messages that must be displayed and which CancerCodingRules must be retrieved and applied.

Each CIARule is uniquely identified by a string in the form "<prefix acronym of change type>-<suffix acronym of CIARule type><number>". The prefix acronym is one of the following:

- DataChange is "DC";
- DomainKnowledgeChange is "DKC"; and
- CancerCodingRuleChange is "CCRC".

The suffix acronyms are 'D' for dependency rule, and 'I' for impact rule. All dependency rules are expected to have at least one corresponding impact rule. The two CIARule types form distinct sets in which each rule do not explicitly relate to one another in any way. This means for instance that rules from the dependency set do not expect a specific order of execution neither from other dependency rules nor from any impact rules. However, there are implicit relations between dependency rules and impact rules; for each object returned by a dependency rule there is an expectation that some impact rule applies (i.e., one that requires the same conditions as the dependency rule in question). Otherwise there would be no impact displayed for that object and there would be no reason for it to be returned by the dependency rule.

CIARules for Data Changes

Dependency rules	**CIARule ID:** DC-D1 Change subtype(s): FieldValueChange **Concept type:** CancerMessage **Description:** A data change will affect the CancerCase *C* to which the changed CancerMessage *M* is related. This relation is by equal cancerCaseNumbers (*C*.cancerCaseNumber = *M*.cancerCaseNumber). Thus, this rule must return a set with only one object: the related CancerCase.	**CIARule ID:** DC-D2 Change subtype(s): FieldValueChange **Concept type:** CancerCase **Description:** CCRs (CCVRs, CFVRs and MARs) that constrain the changed field must be retrieved so that they can be applied by impact rules.	**CIARule ID:** DC-D3 Change subtype(s): FieldValueChange **Concept type:** CancerMessage **Description:** CCRs (MVRs and CFVRs) that constrain the changed field of CancerMessage *M* must be retrieved so that they can be applied by impact rules.
Impact rules	**CIARule ID:** DC-I1 **Change subtype(s):**	**CIARule ID:** DC-I2 **Change subtype(s):**	**CIARule ID:** DC-I3 Change subtype(s): FieldValueChange

	FieldValueChange **Concept type:** CancerMessage **Description:** The changed CancerMessage *M* must be re-validated and re-aggregated in case of invalidation by any constraining CCRs. This impact rule should display a notification that the message must be revalidated. Any MVRs and CFVRs constraining the changed field must be applied to the changed object to re-validate the impacted CancerMessage. The result of the application (whether or not the change caused the rule to invalidate the message/case) should be displayed.	FieldValueChange **Concept type:** CancerCase **Description:** If the changed field of the changed CancerMessage *M* is a CommonField, the value of the same field of the related CancerCase *C* will be changed as well. In any case, *C* must be revalidated in case of invalidation by any constraining CCRs. Any CCVRs and/or MARs constraining the changed field must be applied to the impacted CancerCase *C*. The result of the application (whether or not the change caused the rule to invalidate the message/case, or if aggregation caused *C* to be changed) should be displayed.	**Concept type:** CCR (any subtype) **Corresponds to dependency rules:** DC-D2; DC-D4 **Description:** The related CCR is not affected, but should be displayed to show that it was among the rules that were applied to the changed object. **Corresponds to dependency rules:** DC-2; DC-D3

Table 3-1: All currently defined CIARules for the data change class of changes in the CRN domain.

Table 3-1 gives the six current CIARules for the data change class. DC-D1 illustrates the most basic of all dependencies in the CRN domain: the dependency of a cancer case on its related cancer messages. When the requested change type is a data change and the changed object is a cancer message, this rule would return the cancer case related to that message. Given the same condition satisfied, the impact rule DC-I1 would analyze the impact of the change to the message by applying any validation rules that constrain the field to be changed; DC-D3 would retrieve those rules. Similar CIARules for the related cancer case are given by DC-D2 and DC-I2; for most change types a CIARule for the cancer message class has a mirroring CIARule for the cancer case class.

The distinction between relations and dependencies in this framework can here be provided an example: consider cancer messages again. They are related to a cancer case by the value of its cancerCaseNumber attribute; other cancer messages that share this relation will also exist. Thus, those cancer messages can be said to be related to each other as well. However, a change to either one of those is not known to ripple to the other messages—they do not depend on each other. Therefore, a dependency rule between cancer messages relating to each other has not been defined. Still, in some cases it would be necessary to retrieve them because

an aggregation rule might be required to be applied (this retrieval task has been left to the impact rules, but an implementation is allowed to use a dependency rule for it if so desired).

Data changes are simple and of low complexity in terms of sizes of returned sets and time use of the queries performed to retrieve those sets. This is fortuitous, as they occur on a daily basis in the CRN.

CIARules for Domain Knowledge Changes

Dependency Rules	**CIARule ID: DKC-D1** Change subtype(s): FieldChange (FieldUpdate OR FieldRemoval) **Concept type:** CancerMessage **Description:** CancerMessage objects that have a non-null value of the removed or updated field in class CancerMessage will be affected. The changed field must NOT be a CommonField. **Other Conditions:** This rule is only valid on the root node, as it would return duplicates on successive calls (i.e., it should be applied only once). It is unfeasible to return all CRN data objects in the database; a single object representing the affected class is sufficient.	**CIARule ID:** DKC-D2 Change subtype(s): FieldChange (FieldUpdate OR FieldRemoval) **Concept type:** CancerCase **Description:** CancerCase objects that have a non-null value of the removed or updated field in class CancerCase will be affected. The changed field must NOT be a CommonField. **Other Conditions:** This rule is only valid on the root node, as it would return duplicates on successive calls (i.e., it should be applied only once). It is unfeasible to return all CRN data objects in the database; a single object representing the affected class is sufficient.	**CIARule ID:** DKC-D3 Change subtype(s): FieldChange (FieldUpdate OR FieldRemoval) **Concept type:** CancerCase OR CancerMessage **Description:** If the changed field is a CommonField, all CancerMessage AND CancerCase objects that have a non-null value of the removed or updated field will be affected. **Other Conditions:** This rule is only valid on the root node, as it would return duplicates on successive calls (i.e., it should be applied only once). It is unfeasible to return all CRN data objects in the database; a single object representing the affected class is sufficient.	**CIARule ID:** DKC-D4 **Change subtype(s):** FieldChange (FieldAddition) **Concept type:** CancerCase OR CancerMessage **Description:** The class to which the new field is added may require the updating of all instances of the class. If the field is a CommonField, both classes are affected. Additionally, new CCRs constraining the new field may be required. This rule does not require implementation, as it applies for the class as a whole and need not fetch any objects.

	CIARule ID: DKC-D5 **Change subtype(s):** FieldChange (FieldUpdate OR FieldRemoval) Concept type: CancerCase OR CancerMessage **Description:** Any CCRs constraining the updated or removed field will be affected.			
Impact Rules	CIARule ID: DKC-I1 **Change subtype(s):** FieldChange (FieldUpdate OR FieldRemoval) **Concept type:** CancerCase OR CancerMessage **Corresponds to dependency rules:** DKC-D1, DKC-D2, DKC-D3 **Description:** Each affected concept object must be displayed as either having the field updated or removed.			**CIARule ID:** DKC-I4 **Change subtype(s):** FieldChange (FieldAddition) **Concept type:** CancerCase OR CancerMessage **Corresponds to dependency rules:** DKC-D4 **Description:** The implementation of this impact rule must display a message stating that the class to which the new field is added may require the updating of all instances of the class, and new CCRs constraining the new field may be required (see DKC-D4).
	CIARule ID: DKC-I5 **Change subtype(s):** FieldChange (FieldUpdate OR FieldRemoval) **Concept type:** CancerCodingRule **Corresponds to**			

	dependency rules: DKC-D5 **Description:** The rule constraining the updated or removed field will be obsoleted. The implementation of this impact rule must retrieve the correct CCRules (i.e., those that constrain the changed field of the class) and notify the user that it has become obsolete.			

Table 3-2: All currently defined CIARules for the domain knowledge class of changes in the CRN domain.

Table 3-2 shows the current CIARules for the domain knowledge change class. Notice that these rules include subtypes of changes. Recall from section 3.3.2 that the ways in which the domain model can change that are supported by the framework includes additions, removals, and updates (removal and addition) to the names and types of fields in the cancer message and cancer case database schemas. In general, the CIARules for the domain knowledge change class center around the obvious fact that objects that have non-null values of the changed fields would be impacted by such changes. However, it would not be feasible to actually retrieve every single cancer case and cancer message with a non-null value of a changed field. If that field were a common field, it would imply that the entire CRN database would have to be retrieved. In reality, field updates do not occur. More likely, a change to a field would concern its *semantics*, in which case what we really want is a change to the cancer coding rules constraining that field. If a field removal were actually requested, it would rather be "deactivated" so that its value no longer holds any meaning (impact rule DKC-I5 in Table 3-2 would likely still apply in such a case). The only change of the domain knowledge class that is likely to occur is the addition of fields, which in and of itself causes no impacts to any persistent data objects or rules; it would only imply that more cancer coding rules be created to constrain that field (again, what we rather want is a cancer coding rule change).

Despite the apparent lack of use for domain knowledge change analysis, it has been requested by the CRN to remain a part of the CIA approach framework for possible future application.

CIARules for Cancer Coding Rule Changes

Dependency Rules	**CIARule ID:** CCRC-D1 **Change subtype(s):** RuleAddition **Concept Type:** CancerCodingRule **Description:** The rule with identical ruleNumber (if any) would be affected, as it would be replaced. **Other conditions:** Must only be applied on the root node.	**CIARule ID:** CCRC-D2 **Change subtype(s):** RuleAddition **Concept Type:** CancerCodingRule (MVR) **Description:** Any CancerMessages of which the diagnosis dates fall within the diagnosis dates of the added rule would have to be re-validated using the new rule.	**CIARule ID:** CCRC-D3 **Change subtype(s):** RuleAddition **Concept Type:** CancerCodingRule (CCVR) **Description:** Any CancerCases of which the diagnosis dates fall within the diagnosis dates of the added rule would have to be re-validated using the new rule.	**CIARule ID:** CCRC-D4 **Change subtype(s):** RuleAddition **Concept Type:** CancerCodingRule (CFVR) **Description:** Any CancerCases of which the diagnosis dates fall within the diagnosis dates of the added rule would have to be re-validated using the new rule.
Impact Rules	**CIARule ID:** CCRC-I1 **Change subtype(s):** RuleAddition **Concept Type:** CancerCodingRule **Description:** The replaced CCR *A* would be deactivated on the activation date of the new CCR *B* (i.e., *A*.activationEndDate is set to *B*.activationStartDate).	**CIARule ID:** CCRC-I2 **Change subtype(s):** RuleAddition **Concept Type:** CancerMessage **Description:** The affected CancerMessages must be re-validated. This impact rule must apply the new MVR/CFVR to each affected CancerCase and display its result.	**CIARule ID:** CCRC-I3 **Change subtype(s):** RuleAddition **Concept Type:** CancerCase **Description:** The affected CancerCases must be re-validated. This impact rule must apply the new CCVR/CFVR to each affected CancerCase and display its result.	

Table 3-3: The currently defined CIARules for the cancer coding rule class of changes.

Table 3-3 gives the CIARules for the cancer coding rule class of changes. CCRC-D1 would return the rule to be replaced if a new rule adopts its unique rule number value. In any case, the dependent objects would be, for each type of cancer coding rule (i.e., message validation rules, case validation rules, common field validation rules, or message aggregation rules), a corresponding set of cancer data objects that would be impacted by this change because they are constrained by the new rule. Whether they are constrained or not is currently defined not only by the type of cancer coding rule, but also by their diagnosis dates. Any data object that

has a diagnosis date that falls within the start and end diagnosis dates of the cancer coding rule would be subject to its constraint. Possibly, more conditions may apply that have not yet been defined here, such as whether or not the field value is null (which means that the particular source of the message either does not record that field, or has made an error). Currently, the assumption is that it is best to apply the new cancer coding rule one too many times than one too few.

3.5 The CIA Algorithm

The final component of the proposed CIA approach framework is the algorithm that computes the EIS. It is the impact/trace technique of our approach. It proceeds along the paths of a change rippling from the changed CRN concept object to dependent objects. The algorithm applies the dependency rules and impact rules on a given change request, repeating itself for each dependent retrieved by the dependency rules. This continues until no concept object could be found by any dependency rule. This section will further explain this three-step recursive algorithm, and provide a proof of its correctness along with theoretical complexity analyses.

3.5.1 Definitions

The algorithm accepts as input an *impact node.* The impact node is a tuple of a CRN change *c*, CRN concept object *o*, an impact *I*, and a set $d = \{vdep_{i0}, \ldots vdep_{n-1}\}$ of adjacent impact nodes. Upon termination the algorithm returns a *graph* $g = \{v, e\}$ in which each vertex v_i is an impact node, and each edge e_i is a path to v_{i+1}. A key observation is that a CRN object should never be impacted more than once by the same change. Thus, each impact node has at most one path to itself, i.e., it has an *indegree* of at most 1, but can have more than one path *from* itself—an *outdegree* of between 0 and n-1 where n is the total number of impact nodes. This implies that the resulting graph is at all times *acyclic.* Acyclic means that there is exactly one path from any given node to another *and* that no node can find a path back to itself. Due to the acyclic nature and the weak connectedness of the resulting graph, it is more precisely described as a *tree.*

3.5.2 Procedure

The pseudocode for the algorithm is shown below:

Algorithm *analyzeImpacts*(impactNode *v*)*:*

1. **FOR EACH** dependencyRule $D_i \ldots D_{n-1}$, do *applydependency rule*(*c, o* of *v):*

   FOR EACH concept object $Odep_i \ldots Odep_{n-1}$ in the set returned by D_i:

   impactNode $vdep_{i+1} \rightarrow createImpactNode(Odep_i)$; add $vdep_{i+1}$ to *d* of *v*

2. **FOR EACH** impactRule $R_i \ldots R_{n-1}$:

   do *applyimpact rule*(*c, o* of *v*); add the impact returned by R_i (if any) to $vdep_{i+1}$

3. **FOR EACH** impactNode $vdep_i$ in *d* of *v:*

   do *analyzeImpacts*($vdep_i$)

In step 1 of *analyzeImpacts*, the algorithm applies each dependency rule (i.e., executes its function) on the concept object and change found in the given impact node. Recall that the rule application will return a (possibly empty) set of CRN concept objects that are dependent on the object given to the rule. Each object of set returned by each dependency rule is added to the tree in a new node. Thus, the result of step 1 is a set that is the union of all sets if concept objects found to be dependent on the given object. In step 2, each of these objects is given its purpose for being in the tree as the impact rules perform their functions. In step 3, for each adjacent node of the current node, if any more impact nodes are connected to the current node (meaning that at least one dependency rule returned a set of at least one concept object for this node), the procedure will repeat itself starting from step 1 using the adjacent node as parameter. When no more nodes exist in this list, the procedure returns. The algorithm terminates when the procedure has returned for each node that was added to the tree. This tree now represents the EIS as its nodes each contain all impacts returned by the impact rules for each concept object determined by dependency rules to be impacted by the given change.

3.5.3 Illustrative Example

As an example to clarify the above described algorithm, consider a case of analyzing a change to a cancer coding rule. Specifically, the example change concerns the replacement of a validation rule. The old rule, with the fictional ruleNumber V99, is specified as follows (this is the same example MVR shown in section 2.2.4):

morfologi = '1527/39' and messageType in['K', 'R'] implies topografi startswith('44').

We wish to change it so that the relevant values of the left-hand side variable "messageType" is only 'K' such that the new rule definition is

morfologi = '1527/39' and messageType = 'K' implies topografi startswith('44').

As for the other attributes of the new rule, the relevant ones are currently the values of the fields diagnosisStart and diagnosisEnd, and activationStart. Recall that the dependency rules CCRC-D2 and CCRC-D3 (see Table 3-3: The currently defined CIARules for the cancer coding rule class of changes. in section 3.4.3) imply that any cancer data objects that have diagnosis dates between within the diagnosis period of the added cancer coding rule would be impacted and thus retrieved. The new rule would be brought into service on its activation date, on which the old rule would be replaced (this is stated by impact rule CCRC-I1, Table 3-3: The currently defined CIARules for the cancer coding rule class of changes.), but can still constrain cancer data objects from before that (the diagnosis period is independent of the activation start, but the activation period cannot end before the diagnosis period[6] of course).

Procedure Call 1

The input to the initial procedure run of the algorithm is the old cancer coding rule and cancer coding rule change (from the new requested rule can be retrieved). Step 1 will apply the dependency rules CCRC-D1 through CCRC-D3, parameters being the cancer coding rule and change in the root node. As the current concept object type is a cancer coding rule (the old one to be replaced), CCRC-D1 returns a set that contains the rule to be changed. A new impact node is then created with this rule as its concept object (this first node is referred to as the root node), and added to the adjacency list of the root node. In the next iteration of step 1,

[6] This particular business rule is not currently reflected by the CIARules. It is a matter of discussion whether to implement it as a CIARule or as an input constraint in the application built on top of the proposed framework. The latter may be more efficient and user friendly, as the application would not have to run an analysis with a change request that could make no sense.

CCRC-D2 would apply and return a set of dependent cancer message objects because the old and new cancer coding rules are MVRs. Step 2 then applies the impact rules, and in the first and only iteration CCRC-I1 adds its impact for the impacted cancer coding rule to the root node. In step 3, the algorithm begins iterating over the set of impact nodes adjacent to the root node. The procedure calls itself, taking the first impact node in this list as parameter.

Procedure Call 2 to *n*

In step 1 of the second call to the procedure, the concept object in the current impact node is one of the cancer messages that are dependent on the changed cancer coding rule. As no dependency rule states that any concept objects would be dependent on a cancer message given when the change is to a cancer coding rule, only empty sets are returned by CCRC-D1 through CCRC-D3. In step 2, however, CCRC-I2 would apply to this node; the new cancer coding rule would be applied to the dependent cancer message, the result of which is recorded in the impact. For step 3 through n, where n is the number of adjacent impact nodes, the next call would cause the same impacts as the current procedure call because all the remaining dependent concept objects are also cancer messages. After each subsequent procedure returns, the algorithm terminates.

Characteristics of the EIS Tree

The EIS tree tends to be *wide* rather than *deep.* I.e., its number of levels, its *depth,* is expected to be far lesser than its number of outgoing paths per node. In the above example, the depth at most reaches 1, while the first node containing the changed cancer coding rule has a path to several cancer messages. Figure 3-2 illustrates the tree resulting from the above example.

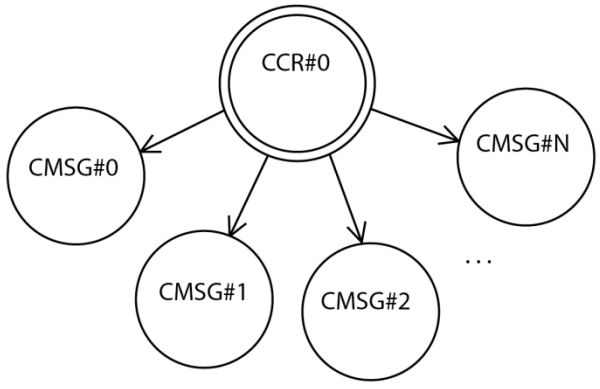

Figure 3-2: Example of a typical EIS tree resulting from the algorithm.

This characteristic currently holds true for all types of changes, but deeper trees may be produced when the algorithm is given more CIARules. This would be expected to happen with cancer coding rule changes because is it for this class that the most rules are still pending implementation (and even more could be discovered).

The algorithm itself is uncoupled from the semantics of the CIARules; it makes no assumptions about their implementation, their numbers, or in which sequence they occur in the rule sets. Thus, it does not impose an ordering constraint on the nodes of the graph. If it is desirable to make the structure searchable, an implementation may use a sorting algorithm, or use auxiliary data structures in which to store the impacts.

3.5.4 Correctness of the Algorithm

The algorithm is considered *correct* when it terminates, in which case it results in a tree representing the EIS. An unsuccessful execution would result in no termination and thus no tree due to either infinite recursion or an infinite loop (depending on implementation details). Not discussed here is the correctness of the EIS, i.e., whether it contains only objects that exist in a given AIS. The algorithm only claims to compute estimation.

For infinite recursion to occur, *both of two* conditions must be satisfied by the dependency rules executed by the algorithm. The first condition is that there is some CIARule that, given one type of change, returns an object that already exists in the tree with the same type of change in its node (a duplicate). Second, there must be a dependency rule stating that the duplicate object has further dependencies such that the algorithm will run additional procedures of itself when given that object. In other words, this would produce a circular dependency. Note that both conditions are likely to be satisfied by the same rule, but there could be exceptions that would require additional dependency rules. Hypothetical (and realistic) examples of dependency rules that would satisfy both of the conditions required for a circular dependency could exist for the cancer coding rule class of changes. First, there could be a rule that returns additional cancer coding rules when given a cancer coding rule (regardless of type). If there is no constraint on the returned cancer coding rules that preclude duplicates, both conditions could be satisfied. If there such a constraint, CCRC-D2 or CCRC-D3 could still cause problems as they could return additional cancer data objects depending on one or more of those cancer coding rules. In this set, duplicate objects could exist if the diagnosis periods of the cancer coding rules in the tree overlap with each other. Then, a rule stating that additional objects depend on the added cancer data objects would have to exist in order for the recursion to continue.

It can be shown that, for the current set of CIARules, the algorithm will always terminate. An informal proof to this is based on the fact that there are no dependency rules that return

duplicate objects for any type of change. I.e., no current dependency rule satisfies the first condition for circular dependencies. For all types of changes, all dependency rules return sets of distinct types of objects when given a certain change type so there cannot be any duplicates across them. Importantly, no rule can be applied more than once for a single change. For cancer coding rule changes, the second condition cannot occur, because the recursion would end with any cancer message or cancer case since there are no defined dependencies on those objects. The recursion ends as CCRC-D2 or CCRC-D3 (Table 3-3: The currently defined CIARules for the cancer coding rule class of changes.) would return only cancer data objects. Similarly, the trivial case of data changes ends with the cancer coding rule objects returned by DC-D2 or DC-D3 (Table 3-1).

3.5.5 Complexity Analysis

Having presented that the algorithm (currently) works, it is worthwhile to analyze its *complexity* in terms of computing time so as to determine if it will terminate within some feasible time period given any type of change. It is also useful to see if this time usage correlates with space usage in terms of the use of a computer's primary memory. As the exact time and space usage measured in milliseconds and bytes respectively would depend on a specific implementation when running on a specific machine, the analysis is limited to a theoretical one. It is useful to at least see a theoretical upper bound on the growth rate of the resource use of the algorithm in a notation that can be used as a basis for comparison with other algorithms. The conventional way of presenting the complexity of an algorithm in such a way is the *Big-O notation.* The 'O', read as "order of", is defined as follows [25]:

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$

$T(N)$ is a function representing our algorithm, and $f(N)$ is a function that represents the upper bound of growth we claim for $T(N)$. Intuitively, the above definition states that $T(N)$ does not grow faster than $f(N)$. We claim that $f(N)$ is a linear function and that $T(N) = O(N)$. N is chosen to be the total number of concept objects returned by the dependency rules, as the algorithm can proceed only as long as there are objects to add to the tree. The proof to the linear claim is by contradiction: because the algorithm cannot execute more than N recursive calls to its procedure, there will always be a c such that $T(N) \leq cf(N)$ when $N \geq n_0$. In less formal terms, this means that the time usage of the algorithm is no more than linear on the order of the total sizes of the sets returned by the dependency rules executed for any given

type of change. The space usage scales proportionally and amounts to the total memory required to store the retrieved objects.

This analysis does however not give the total complexity of the algorithm as it is actually the dependency rules that do most of the work; they are queries that must be executed by some search algorithms in order to retrieve the desired objects. Thus, what would be required to complete the complexity analysis is an analysis of each of the dependency rules. Indeed, the actual complexity of our algorithm should be a function of the dependency rules, i.e., O(*N*) times the total time and space usage of all the dependency rules. In other words, we would be more interested to see its complexity when *N* is the number of total available concept objects in the CRN. These analyses are however impossible as we do not yet know what search algorithms will underlie the specific implementation of the approach framework. We also do not have an estimate of how many objects could be retrieved in the worst case by each of the rules (except for the base cases that find the root node). Hence, a theoretical analysis cannot answer the question "will the algorithm terminate within a feasible amount of time while using a feasible amount of space". The empirical evaluation of the approach will look at the worst-case growth of the EIS. The desired algorithm complexity is an upper bound EIS of the size of the AIS for any given change type (of course, the lower bound should be as close to the upper bound as possible).

3.6 A Prototype Implementation of the Proposed CIA Approach

This section presents the prototype software tool that has been developed for the purpose of evaluating the approach framework. It consists of a prototype CIA application programming interface (API) based on the approach framework, with a desktop application built on top. The API implements the CRN domain model, the change model, the CIARules, and the CIA algorithm. It is developed in Java 8, and the application is built with the JavaFX API. Java 8 was chosen due to its support for object-oriented programming and functional programming. The former is important due to the heavy reliance of the approach on classification models; the latter is interesting as it allows the representation of functions as objects, which simplifies the implementation of cancer coding rules and the CIARules. The tool supports all change types using the current CIARule set (see 3.4.3); a graphical user interface (GUI) is provided by the application for inputting changes and to view results of the analyses. All analysis

results include lists of impacted concept objects (output of the dependency rules) with the impacts (output of the impact rules) listed for each concept object. Additionally, performance metrics are displayed. These include run time measured in milliseconds, and precision and recall when an AIS is provided.

3.6.1 CRN Domain Model and Change Model Implementation

The prototype API implements the models manually, i.e., it does not use a model-based tool for generating code from models. This was done due to the developer's skill level with any such tools; it was decided that it could be done faster using Java. The implemented CRN domain model is true to the original model in that its classes contain only the attributes found in the original model, but it does not include the PatientHistory class, as this has yet to be proven necessary to exist for any CIARules. Furthermore, another minor adaptation to the domain model is that the CommonField class has been represented as a superclass of the CancerMessage and CancerCase classes, rather than being part of a compositional relationship. The implemented change model includes at least all the classes and their attributes of the original model as presented in 3.3.

3.6.2 CIARule and CIA Algorithm Implementation

The CIARules are implemented as *functional interfaces*, which are classes that represent functions. The implemented CIARules can be expected as behaving as specified in section 3.4.3; no further details are required to understand what they do within the prototype. Cancer coding rules are represented in the same way. The dependency rules are given access to cancer data objects via an in-memory object index. The prototype does not currently operate on actual data from the CRN databases, as that would require access privileges not afforded the developer.

3.6.3 The Prototype Application

The application provides the user with functionality for analyzing the three main types of changes. The overall task flow is the same for all use cases and follows three main steps: (1) the user selects the type of change; (2) the user inputs the requested variables for the change; and (3) the user confirms the change after which the analysis is run and the results are displayed. Thus, the application satisfies the automation requirement (see section 3.1.1)—

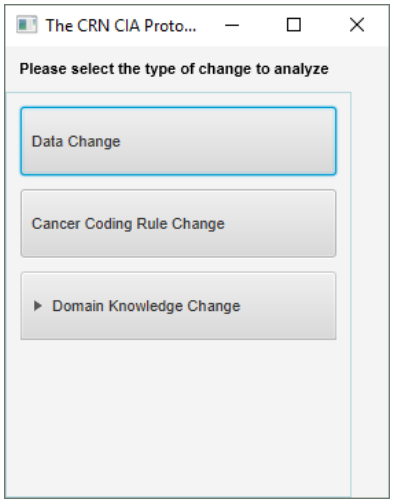

Figure 3-3: The main change type selection menu of our prototype tool.

only the change request must be input manually by the user. Figure 3-3 shows the main menu of the tool from which the user selects the change type in step (1). The application currently only supports the analysis of one change at a time, i.e., there is no way to queue up a sequence of changes. The tool only stores copies of data so that it cannot corrupt real data when simulating changes. Actually, the tool does not even change its own data that; it applies changes only to clones of objects.

Data Change Functionality

To analyze a data change, the first step is to select "Data Change" from the main menu on the left side of the window. Then, a list of all cancer message objects is shown in the middle section. To proceed, the user must click one of these objects to expand it and show its fields. Then, a field to change must be selected, upon which the user is prompted to choose a new value of the field. Figure 3-4 shows this view after a field has been selected for change and the input dialog has appeared. After entering the value, which has to correspond to the type of the field (e.g., an integer field must have a value that contains only numeric characters), and confirming the change, the analysis is run and its results are shown on the right side. All concept objects impacted by the change according to the dependency rules are shown. Clicking on one will expand it and show the impacts for this object as determined by the impact rules. Figure 3-5 shows this view with a selected impacted object. Here, a retrieved cancer coding rule is selected, and displayed is its reason for being in the impact list.

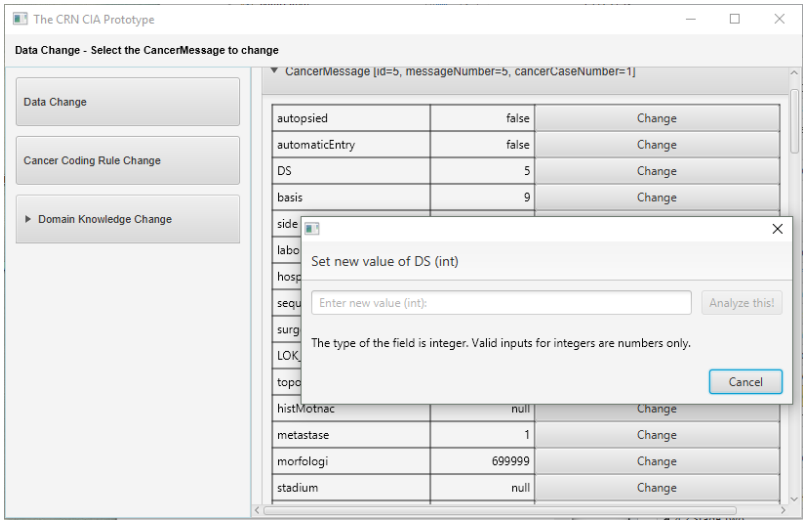

Figure 3-4: View of the data change input dialog.

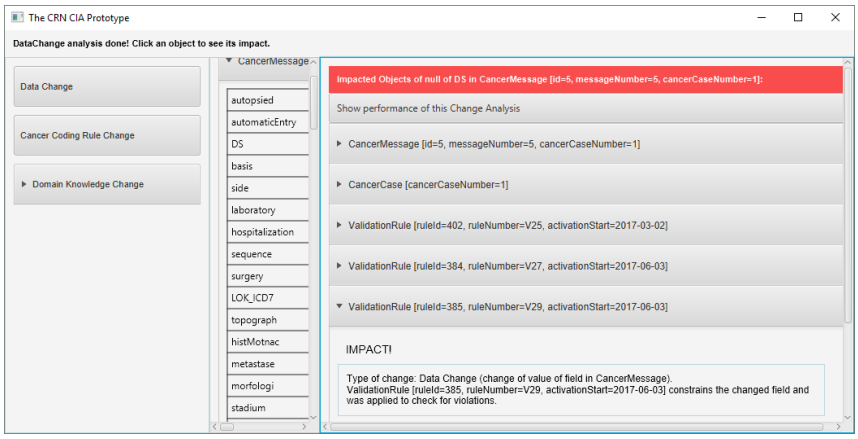

Figure 3-5: The data change impact list after changing a cancer message value.

Domain Knowledge Change Functionality

For domain knowledge changes, the user must also select from the main menu the class that should be changed, either CancerMessage or CancerCase. This will display all the fields for the chosen class, one of which must be selected either for changing (a FieldUpdate) or for deletion (FieldDeletion). A third option supported is the addition of a new field (FieldAddition). For field additions and field updates, the user must input a new name and a new type for the field.

Cancer Coding Rule Change Functionality

When changing a cancer coding rule, the user must actually first create a new one. Currently, validation rules are supported for change analysis through the application GUI (the API supports MAR changes, but currently lacks CIARules for them). The change type would be determined by the existence of a cancer coding rule with the same rule number as the new one; if one exists, the change is regarded as a RuleUpdate; if not, it is a RuleAddition. The input parameters for the new rule are the dates that start and end the diagnosis and activation periods, the rule number, rule ID, rule type (general or exception), the type of cancer data objects that the rule will constrain, and an OCL expression that specifies the rule function itself. The API will automatically generate a function object for the new cancer coding rule from this expression so it can be applied to any objects that are constrained by the rule. It is required that the OCL expression is well-formed according to the OCL syntax, and that the fields constrained by the rule exist in the class specified by the validation rule type (i.e., MVR

for cancer messages, CCVR for cancer cases, and CFVR for common fields). If all conditions are satisfied, the results will be displayed on the right hand side as usual.

Performance Measurement Functionality

On top of the analysis results list, a button labeled “Show Performance” can be seen. Clicking this button will bring up the performance numbers of the previous change analysis. It is available for all change types but is not meaningful for domain knowledge changes. The numbers include (1) run time, (2) numbers of cancer cases changed, (3) numbers of invalidated cancer message and cancer case objects, (4) the percentage of fields changed across all those cancer cases, and (5) the precision and recall measurements (if an AIS was provided for the analysis). The precision and recall metrics are of special interest when evaluating the approach.

This chapter has presented the proposed CIA approach framework and a software prototype based on this framework. The framework incorporates the CRN domain model and a custom change model for representing the application domain. Changes are analyzed according to change impact analysis rules that are specific to the CRN. The framework supports a software implementation that can automatically retrieve impacted objects, and for each of those objects show what would happen to them if the change were applied. We presented a software prototype implementation of the framework that can accept and analyze user-defined changes of all main types in the change model.

4 Evaluation of the Approach

This chapter describes the final phase of the project: to evaluate the proposed approach. It is emphasized that the evaluation will focus on the performance of the *approach*, and not on the performance of the software prototype tool. The evaluation process has been divided into two stages. Stage one has the goal of gathering feedback from CRN stakeholders in order to assess the applicability of the approach framework in the context of the CRN. Given implementations of the requirements established in cooperation with the CRN in stage 1, stage two focuses the empirical verification of the correctness of the currently defined CIARule set as presented in section 3.4.3.

4.1 Stage One

This section accounts for the design and execution of the first of the two stages of the evaluation phase of the project.

4.1.1 Research Questions

This first stage aims to address the following research questions:

1. RQ1: To what degree does the proposed approach framework support the actual business process of changing concepts in the CRN?

2. RQ2: For each change type currently supported, does the approach produce the expected output in terms of (a) impacted objects, and (b) the impact descriptions for impacted object.

3. RQ3: What other functionality could be desired or expected by the potential end-users of a CIA solution in the CRN?

RQ1

RQ1 arises from the uncertainty surrounding the process of changing in particular (but not limited to) cancer coding rules. What was known about changing any type of CRN concept at this stage had been learned from the study of the MBE4CR works (i.e., [12, 13]), which provided the basis for defining an initial CIARule set for all change classes. However, as the

dependency rules require explicit knowledge about relations between objects, and relations between cancer coding rules at this point were uncertain, the confidence in the dependency rules currently defined for the cancer coding rule change class was low. Simultaneously it was decided to take advantage of the opportunity to verify the knowledge about the other two main classes of changes. In other words, the purpose of asking this question was to validate the change model. After all, if the change model inaccurately represents the real world, the CIA approach could not accurately simulate real-world changes and thus predict their impacts in any meaningful way.

RQ2

Validation of the current set of CIARules was necessary for the same reason as for validating the change model. However, one key consideration is that, even given a correct change model, the approach would still not give the expected output if the CIARules were incorrect—after all, they provide the output. Thus, it is desirable to validate the CIARules separately. If the dependency rules retrieve objects that should *not* be impacted for any given type of change, users' confidence in the CIA tool could rapidly be lost. Conversely, if the dependency rules would not retrieve objects that should have been impacted, the result could be overconfidence in the tool, i.e., users may believe that it is doing its job when it is in fact not.

The consequences of incorrect impact rules are somewhat more difficult to define. A message describing an impact could be correct, incorrect, partially correct, or correct but difficult to understand for the user due to poor wording. Then, what is "correct" or "poor" could be subjective to the particular user. Thus, the feedback of interest here would rather be what *types* of information the user would expect given some impact of some change. The main concern of the impact rules is that they provide useful feedback, such as why an object was related to the changed object, and what could happen to it if the change were to be applied in the real world. Thus, the purpose of soliciting feedback on the current impact rules is to improve the usefulness of their returned impact descriptions.

RQ3

The third research question arises because it is expected to be answered even without asking it; the CRN stakeholders would undoubtedly reveal a wish list of functionality as soon as they

are informed of the capabilities of the approach. Furthermore, feedback on the functionality of the current version of the prototype tool is also expected and welcomed. The feedback regarding functionality would be used for improving the approach prototype for future evaluation steps. It could also bring to light the need for additional functionality support on the framework level.

4.1.2 Method

The method selected to investigate the above research questions are group interviews with CRN domain experts, including participants from the MBE4CR project (see section 2.2) with Simula who work closely with the CRN, and one primary CRN stakeholder (the head of the Registry Informatics Department). A total of three such sessions were held over the course of this project phase. Each session centered on a presentation of the approach using the prototype tool presented in 3.6, running through a predefined set of use cases. Each use case analyzed one change type, showing the steps and input required to do so. The focus of the interviews was not so much to ask for feedback, but more about *eliciting* feedback. I.e., the participants were encouraged to interrupt and provide corrections if they were to witness some unexpected output (or lack of expected output) or any other inaccuracies in the processes of the demonstrated use cases. Afterwards, they were asked to openly comment further on the approach and any aspect of the prototype if so desired. The overall session procedure was as follows:

1. Present a use case of a change while describing the procedure.
2. Explain the output (i.e., why and how the objects were impacted) displayed on screen after analysis of the change.
3. Take note of any feedback elicited under the demonstration.
4. Repeat procedure from step 1 if there are use cases left to demonstrate.

An important part of step 2 in the above procedure is to explain the CIARules that were applied in order to produce the output. This would explain to the participants how the approach works, and likely elicit any feedback on its correctness with regards to the real-world expectations.

For the first interview session of the evaluation phase, the approach design would of course have to be explained before demonstrating it. Participants were given a high-level description of its conceptual framework, with demonstrations of example change analyses using the prototype tool. Use cases for the change classes DataChange and DomainKnowledge were available in the first session. New CIARules for CancerCodingRuleChanges had to be defined and implemented based on feedback from this session, as there were too many uncertainties surrounding them currently. Thus, use cases for CancerCodingRuleChanges could not be tested until sessions two and three.

4.1.3 Results of Evaluation Stage One

Overall, the tool appeared to not produce any unexpected output according to the participants; the dependency rule set was deemed correct, but currently lacking in coverage of impacted objects. The feedback gathered enabled the definition of several more CIARules, in particular for the CancerCodingRuleChange class. Additionally, some improvements were made to the current set of DataChange impact rules; the participating CRN stakeholder highlighted the importance of being able to see results of applied cancer coding rules to changed cancer data. This means that the application would have to be able to apply these rules to the data, and display the resulting cancer messages and cancer cases. Hence, functionality for applying cancer coding rules, both VARs and MARs, was implemented in the API. Furthermore, a crucial functional requirement established was that for all types of changes, the performance analysis should provide statistics. Specifically, it must show the number of changed cancer cases, and for each of those show the number and percentage of changed field values. This is especially important when analyzing cancer coding rule changes, as these changes could affect the most cancer data objects out of any change type. It would also be useful for data changes, at least when analyzing long sequences of changes.

In summary, this preliminary evaluation led to the conclusion that the proposed framework is applicable in the CRN, but that further improvements to the CIARule set were required. The CIARules and prototype tool were refined consequently, which resulted in the current set as presented in section 3.4.3 and the version of the tool presented in 3.6. Further empirical evidence was however required to verify the correctness of this new and improved CIARule set. In particular, we had yet to measure performance in quantitative terms.

4.2 Stage Two

Whereas stage one of the evaluation phase focused on the more qualitative aspects of the approach, the second and final stage examines the quantitative performance measurements of the approach. The research question for this stage continues to be RQ2, stated in 4.1.1, but is now attempted to be answered rigorously. Performance is measured in terms of precision and recall of the estimated impact sets returned by the prototype tool, in addition to counting the number of cancer data objects that are the subject of invalidation and re-aggregation by cancer coding rules following a change. We hypothesize that (1) the precision of the tool is near-optimal, and (2) that the cancer coding rules are correctly retrieved and applied whenever necessary.

The set of changes to be analyzed is designed in cooperation with domain experts from the MBE4CR project. The AIS containing the expected results of the analyses is also provided by the domain experts to enable the precision and recall measurements. Finally, the analyses were carried out by the experts using the prototype tool, recording results along the way.

4.2.1 Method

We involve two domain experts from the MBE4CR project to assist with this stage of the evaluation. Specifically, they participate in (1) defining a set of changes to analyze with the prototype tool; (2) defining a global system set of CRN concept objects; (3) defining the AIS for each change, and (4) with performing the analyses. The set of changes and the data sets are described in 4.2.2. The experts are trained to use the tool for the analyses. For each change analysis performed, the values measured for the performance metrics given in section 4.2.3 are recorded. Note that the sequence in which the changes are analyzed is immaterial as the tool does not make persistent changes to the objects in its data store. This also allows objects to be reused for further change analyses.

4.2.2 Data Sets

The domain experts assisted with manually creating all data sets accounted for in this section, including the set changes to analyze.

Changes to Analyze

The set of changes is designed to cover two out of the three main types of changes (see section 3.3), including the supported change subtypes. Table 4-4 gives the changes to analyze in this evaluation.

Change Type	**Change Parameters**				**AIS#**
	Change ID	Object ID	Field Name	New Value(s)	
Data Changes	DC1	CM0	basis	61	1
	DC2	CM1	metastase	"D"	2
	DC3	CM6	diagnosisDate	01-01-1993	3
	DC4	CM3	radiotherapy	0	4
	DC5	CM4	hormoneTherapy	9	5
	DC6	CM5	messageType	"O"	6
	DC7	CM2	chemotherapy	1	7
	DC8	CM7	surgery	0	8
	DC9	CM0	morfologi	"690089"	9
Cancer Coding Rule Changes	CCRC1	373	basis	{375, V25, 31-12-2018, "DS ='5' implies Basis in['00', '10', '20', '29', '30', '31', '40', '41', '71']"}	10
	CCRC2	N/A	N/A	{001, V1, 01-01-1999, 01-01-2019, "DS in['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']"}	11

Table 4-4: The set of changes to analyze in the evaluation.

The first column from the left in Table 4-4 gives the change type, and the second column gives the change parameters. One change is defined in terms of the CRN concept object to change, the field or value to change (which depends on change type as discussed in section

3.3.2), and the new value with which to replace the old. Changes are given a unique identifier consisting of the change type acronym and an integer; the identifiers of the objects to change follow the same syntax except for cancer coding rules, which use their real rule identifiers ("ruleID", integer). All objects in the system set are given in A-1 in the appendix, and an excerpt is given in Table 4-5. Lastly, the third column gives the identifier of the AIS that is expected of each change; the AISs are given in 4-6. A total of 11 changes are defined and analyzed: nine data changes, and two cancer coding rule changes.

The data changes comprise the largest set of changes because it is the most common type of change to occur in the CRN. Furthermore, the data change set is designed so that at least one cancer coding rule for each constrained field are retrieved and applied by the impact rules. Only one domain knowledge CIARule retrieves any objects (DKC-I5; see Table 3-2), and this class of changes does not represent a unique case of changes because the retrieval and application of cancer coding rules is already exercised by other changes. Therefore, no domain knowledge change is analyzed here. For cancer coding rule changes there are currently two unique cases of changes represented in the CIARules: RuleUpdate and RuleAddition. The change set covers all CIARules by analyzing one of each of those change subtypes: CCRC1 and CCRC2, respectively. CCRC1 adds a rule with a ruleNumber that is already occupied by another cancer coding rule in our system, and attempts to remove two of the implied values of the common field "basis". Also, its diagnosis period is extended to 2019. CCRC2 simply adds a new cancer coding rule that constrains the field "DS", with a diagnosis period between January first 1999 through January first 2019.

System Set

The system set is the set of all the CRN concept objects that could be affected by the given changes. This set is designed to provide the changes with data that will cause all the currently defined CIARules for data changes and cancer coding rule changes (section 3.4.3) to be applied and return results. Figure 4-1 shows the global system set for the cancer message object and cancer case object sets.

Type	Object ID	Field Name and Value											
		cancerCase#	DS	basis	metastase	diagnosisDate	radiotherapy	hormoneTherapy	chemoTherapy	surgery	morfologi	messageType	topography
Cancer Message	CM0	98	5	36	5	07-01-96	0	1	2	0	700001	K	630
	CM1	99	5	60	F	01-01-93	4	1	6	0	690009	K	890
	CM2	98	6	98	0	01-01-16	1	5	2	0	740509	R	500
	CM3	101	1	30	8	01-01-15	3	1	6	0	740009	H	621
	CM4	100	0	40	1	03-01-17	2	2	1	0	908019	O	623
	CM5	1	5	9	1	01-01-17	6	7	3	2	699999	O	0
	CM6	1	1	10	0	01-01-16	2	9	0	0	50	K	0
	CM7	1	5	30	D	02-01-16	9	8	2	1	30	R	0
Cancer Case	CC0	100				01-01-16							
	CC1	101				31-12-16							
	CC2	99				01-01-93							
	CC3	98				01-01-93							
	CC4	1				01-01-97							

Figure 4-1: The cancer data objects subset of the system set used in the evaluation.

Eight cancer message objects and five cancer case objects are included for a total of 13 cancer data objects. The fields that are used in the objects are those that are constrained by the cancer coding rules included in the evaluation; an excerpt of the cancer coding rule system subset is given in 4-5. As we only have an aggregation rule for the diagnosisDate field, the other fields are null (their values have not been aggregated). This is accepted because it should invoke violations by cancer case in the cancer coding rule changes, which would not be caused by any of the cancer messages.

CCR Type	**Field Name**				**OCL Expression or Function Description**
	ruleID/ Object ID	ruleNumber	diagnosisStart	diagnosisEnd	
MVR	386	V29	01-01-1993	31-12-2016	(Morfologi in['720039', '730019', '730039', '740009', '740509', '741009']and messageType != 'O') implies Basis in ['57', '72', '98']
	387	V30	01-01-2017	Infinity	(Morfologi->startswith('9990') and messageType in ['H', 'O']) implies Basis = '98'
	388	V30	01-01-1993	31-12-2016	surgery = '00' implies messageType in ['K', 'R']
CFVR	360	V16	01-01-1993	31-12-2016	Basis = '98' implies (Morfologi in['340619', '340709', '340809', '341139', '720039','730019', '740009', '740509', '741009', '730039', '750009'] or Morfologi->startswith('9990'))
	361	V16	01-01-2017	Infinity	Basis = '98' implies Morfologi->startswith(

					'9990'))
	329	V19	01-01-1993	31-01-2016	radiotherapy in [null, '0', '1', '2', '3', '4', '5', '9']
MAR	MAR0	A1	N/A	N/A	The value of diagnosisDate is set to the earliest eventDate found in all related CancerMessages.

Table 4-5: Excerpt of the cancer coding rules included in the evaluation system set.

Table 4-5 divides the set of cancer coding rules into the included cancer coding rule types: MVR, CFVR, and MAR. A total of 20 cancer coding rules were included in the evaluation: six MVRs, 13 CFVRs, and one MAR. Table A-1 in the appendix gives the full set. For the MVRs, the OCL expression for the rule is given, and for the MAR function an informal description is given. For the introduction to cancer coding rules, refer to section 2.2.4. The details of the MAR included are not important, other than its function: to aggregate the value of the field diagnosisDate. This MAR should be applied in DC3. Note that all rules included are real rules from the CRN. The ruleID attribute is used in the real world CRN to distinguish between different versions of the same rule, and we use it as the Object ID for rules in our data set tables.

Actual Impact Sets – The Gold Standard

The AIS concept, introduced in 2.1.1, includes, for some change, which objects in the system data set should be impacted given that change. In this evaluation there is one AIS (not necessarily distinct) for each change given in Table 4-4. Table 4-6 gives these sets. The sum of all AISs is referred to as our "gold standard".

AIS#	Change ID	Dependent Object IDs	Invalidated Object IDs	Number of Changed Cancer Cases
1	DC1	CM0; CC3; 357; 358; 360; 361; 373; 402; 386 387 (10 objects)	CM0 (violates 373, 402)	
2	DC2	CM1; CC2; 357; 358 (4 objects)	None	
3	DC3	CM6; CC4; MAR0 (3 objects)	None	1 (CC4.diagnosisDate is changed to 01-01-1993)
4	DC4	CM3; CC1; 361; 400 (4 objects)	None	

5	DC5	CM4; CC0; 365 366 (4 objects)	CM4 (violates 366)	
6	DC6	CM5; CC4; 386; 387; 388; 389 (6 objects)	None	
7	DC7	CM2; CC3; 367; 368 (4 objects)	None	
8	DC8	CM7; CC4; 388; 389 (4 objects)	CM7 (violates 389)	
9	DC9	CM0; CC3; 360; 361; 374; 384; 385; 386; 387 (9 objects)	None	
10	CCRC1	CM0; CM1; CM2; CM3; CM4; CM5; CM6; CM7; CC0; CC1; CC2; CC3; CC4; 373; (14 objects)	CM0; CM1; CM5; CC0; CC1; CC2; CC3; CC4 (8 objects violate the new rule)	
11	CCRC2	CM2; CM3; CM4; CM5; CM6; CM7; CC0; CC1 (8 objects)	CC0; CC1; CC2; CC3; CC4 (5 objects violate the new rule)	

Table 4-6: The AISs with expected cancer coding rule violations and changed cancer cases for each change analyzed in the evaluation.

The third column from the left gives, for each change, the objects that are expected to be retrieved by the responsible dependency rules. Note that, for changes to common fields, cancer coding rules may be retrieved and applied twice: once for the changed cancer message, and once for the related cancer case. The fourth column shows which objects should be invalidated by any cancer coding rules (VARs) retrieved for each change (this is only applicable for changes that invoke re-validation). The fifth column contains the expected number of changed (following re-aggregation) cancer cases where applicable.

4.2.3 Performance Metrics

Recall from section 2.1.1 that a widely used pair of metrics for measuring the performance of a CIA technique is *precision* and *recall*. They have their origin in evaluating information retrieval (IR) techniques, in particular *classification* [26]. Classification is generally a task of assigning some object to some class that is most appropriate based on the features of the object. The task of determining impacted objects can be regarded as *binary* classification, meaning objects are assigned to one out of two possible classes: *impacted* or *not impacted*. This is why the precision and recall metrics are equally useful for evaluating a CIA technique as they are for evaluating a classifier. We choose *P* and *R* as they are trivial to compute and provide an exact quantitative measurement of the level of correctness of the current dependency rule set. “Correctness” here refers to what degree the dependency rules succeed in retrieving the objects that should impacted by a given change with respect to the AIS.

Precision

Precision *P* is informally defined as follows:

$$P = \frac{|EIS \cap AIS|}{|EIS|}$$

P computes the number of objects in the EIS that were also in the AIS, over the size of the EIS. Thus, it gives the fraction of objects that were determined to be impacted that should *actually be impacted.* The formal definition takes into account the number of retrieved true positives (*tp*) and the number of retrieved false positives (*fp*):

$$P = \frac{tp}{tp + fp}$$

Recall

Recall *R* is informally defined as follows:

$$R = \frac{|EIS \cap AIS|}{|AIS|}$$

R computes the fraction of objects determined to be impacted that are actually impacted out of all objects that should be impacted. The formal definition takes into account the false negatives (*fn*):

$$R = \frac{tp}{tp + fn}$$

Ideally, the *P* and *R* measurements would both be 1. A *P* <1 would mean that false positives are found (the current rule set found objects not in the AIS), while an *R* <1 would indicate the need for more dependency rules (the current rule set missed objects in the AIS). The types of objects missed could indicate the change type for which new rules must be defined. Thus, the purpose of measuring *P* and *R* is to validate the existing dependency rules and guide the definition of new dependency rules.

P and *R* results will always be reported together because otherwise they can be misinterpreted. As *P* shows only the fraction of the EIS that was actually impacted, it cannot

tell whether all objects that should have been impacted were found—this is what *R* does. On the other hand, *R* by itself cannot tell whether the EIS covers *more* objects than those in the AIS. This means that even if $R = 1$, there can still be objects in the EIS that are not supposed to be there (which would be shown by $P < 1$).

Re-Validations and Re-Aggregations

Our prototype software tool applies cancer coding rules to impacted objects when the impact rules specifies that it is required, and records and displays the results of the applications in the results view. As a part of the AIS, there will be a certain number of cancer data objects that are expected to be re-validated and re-aggregated when given a certain change. For data changes there will be invalidated cases and invalidated messages, and cases that will be re-aggregated. For cancer coding rule changes there will be invalidated cases and invalidated messages, but no re-aggregations. The quantitative metric to be used for evaluating the impact rules is the number of correctly invalidated cases and messages, and the number of correctly re-aggregated cases. Also, there are cancer data objects in the AIS that should not be invalidated or re-aggregated following any change; any result showing these as the opposite would indicate a problem with the responsible impact rules.

4.2.4 Hypotheses

Having now accounted for our performance metrics, we can state our two hypotheses.

h1

h1 concerns the *P* measurements:

> *The precision P of all changes analyzed is ≥0.95.*

The null hypothesis of h1, $h1_{null}$, is thus:

> *The precision P of no change analyzed is ≤0.95.*

h1 is based on the expectation that the dependency rules can only retrieve objects that are actually impacted, with a margin of error of 0.05. Note that we make no specific hypothesis on the *R* value of any change analysis; it is only expected that it correlates with *P* to some extent. Of course, if $P > 0.0$, *R* must also be >0.0.

h2

h2 concerns the number of correctly identified cancer data object re-validations and re-aggregations:

> *The percentage of applied cancer coding rules that correctly validated, invalidated, and re-aggregated a cancer data object following all changes analyzed is ≥95.*

$h2_{null}$ is:

> *The percentage of applied cancer coding rules that correctly validated, invalidated, and re-aggregated a cancer data object following all changes analyzed is ≤90.*

h2 expects that the impact rules are capable of retrieving and applying the correct cancer coding rules when they are supposed to do so.

4.2.5 Errors

Sources for potential systematic errors in the evaluation method exist in (1) the software tool, (2) the starting impact set defined with the domain experts, and (3) the AIS designed with the domain experts.

Errors from the software could result from defects in the implementation of the CIARules that cause the algorithm to provide either false positives or true negatives that are not a result of the definition of the CIARules. Also, cancer coding rules may not be return the correct results (regardless of whether they were correctly retrieved or not).

Errors from the starting impact set could stem from values of the cancer data objects in the starting impact set that are not *realistic*. Realism here entails that the objects conform to the constraints imposed by cancer coding rules such that the cancer data objects represent real-world data. Dependency rules determine dependencies between objects based on the values of particular fields; if the values are not realistic, there can be no valid proof of correctness of these rules. If in the real world any given rule is not actually correct, then realistic objects in the test set would result in false positives from this dependency rule, which would be verified by the AIS. A randomly generated object on the other hand could produce false negatives that confirm the correctness of the dependency rule when it is in truth incorrect. Unfortunately, actual cancer data from the CRN databases could not be procured for this evaluation due to

safety concerns. The compromise was to involve domain experts in order to create a verifiably realistic data set.

The AIS is of course the data set most likely to cause systematic errors, because every measurement of the metrics given in 4.2.3 assumes that the AIS contains the whole truth and nothing but the truth. If this is not actually the case, then both false positives and true negatives could occur regardless of the correctness of implementation and starting impact set.

4.2.6 Results of Evaluation Stage 2

Table 4-7 gives the *P* and *R* values, and the fractions of correctly re-validated or re-aggregated cancer data objects (where applicable) recorded for each change analyzed.

Change ID	**Precision *P***	**Recall *R***	**Fraction of Correct Retrievals and Applications of CCRs**
DC1	1.0	0.60	1/2
DC2	1.0	0.75	1/2
DC3	1.0	1.00	1/1
DC4	1.0	0.66	1/2
DC5	1.0	1.00	2/2
DC6	1.0	0.66	2/4
DC7	1.0	0.75	1/2
DC8	1.0	0.75	1/2
DC9	1.0	0.77	5/7
CCRC1	1.0	1.0	13/13
CCRC2	1.0	1.0	8/8

Table 4-7: The results of the change analyses in the evaluation.

For each change analyzed, the tool returned an EIS containing only objects that are found in the respective AIS, meaning a precision *P* of 1.0 in all cases. This confirms hypothesis h1, which states that the precision will for every change analysis be ≥0.95. The recall *R* ranged from 0.60 to 1.0, with ~63% (7/11) of analyses producing an R <1.0.

In ~27% (2/11) of the analyses did the tool correctly retrieve and apply the cancer coding rules applicable for the given change; 36 out of 45 (80%) cancer coding rules were correctly retrieved and applied. This disconfirms the hypothesis h2, which states that the tool should correctly retrieve and apply ≥90% of cancer coding rules across all changes analyzed. The reason for this measurement being lesser than hypothesized was later revealed to be caused by a defect in the prototype tool. The issue was that the tool did not correctly index cancer coding rules with a non-unique ruleNumber. I.e., for each cancer coding rule for which there

is more than one version, the tool could only retrieve one of the versions because the others did not exist in the expected coding rule index. Following manual inspection of the retrieved objects for each change analysis, this defect was predictably also shown as the cause of the less-than-optimal measurements of recall—the objects not retrieved were the non-indexed cancer coding rules.

The cancer coding rule changes proved the most complex in terms of EIS size. In CCRC1, the entire cancer data object subset was retrieved, due to the diagnosis dates of the objects all being within the diagnosis period of the changed cancer coding rule. CCRC2 retrieved 8 out 13 objects.

5 Discussion

The evaluation of the proposed CIA approach aimed to prove its applicability in the CRN domain. We evaluated in two stages. The first stage was an iterative process in which a crucial CRN stakeholder was involved to examine the approach framework and its prototype software implementation. The goal of this stage was to validate the currently established requirements of the approach implementation, and to establish new requirements. At the same time, the conceptual framework of the approach was judged; we aimed to determine whether it could support the new requirements proposed, or if we would have to modify the framework in any way. In the second stage of the evaluation, we aimed to further verify and validate the approach on both the conceptual and implementation level using well-established quantitative metrics.

5.1.1 Applicability of the Approach in the CRN

Considering the results of stage one of the evaluation phase, the approach framework can be said to be correct with regards to the real-world CRN domain according to the participating CRN stakeholder and the participants from the MBE4CR project. As the internal model of the approach incorporates the CRN domain model that was developed by this project, the objects that undergo changes in this domain was expectedly deemed as correctly defined. The second part of the internal model, the change model, was based on the domain model and previous research done by MBE4CR on changes. The change model was also deemed as correct with regards to the knowledge of the participants. Regarding the CIARules, following three iterations of the first evaluation stage, the current set was determined by the CRN stakeholder to be correct in the sense of accurately describing impacts to the correct types of objects given a certain type of change. The results of evaluation stage two support the findings of stage one, quantitatively showing a high level of accuracy (precision and recall) of the approach using a prototype implementation on a gold standard hand-made by domain experts. We assume that, if the internal model and the rule set were not correctly defined, the accuracy should not be this high. A characteristic of the approach framework that has been observed is its *flexibility*. It is flexible in the sense that there has so far been no need to alter its concepts in any way; the new requirements elicited from the evaluation imposed only changes on the implementation level. To support newly discovered impacts and dependencies, the only action required is to define and input new CIARules. A wealth of evidence vouching for the applicability of the

approach in the CRN has been gathered, but a disproval that it *cannot* be applied has not yet been achieved.

The above discussion rests on certain implicit assumptions. First, it assumes that the evaluation participants have perfect domain knowledge with regards to the real-world situation. Second, it assumes that all CRN associates who might be using an implementation of the proposed approach in the future share the understanding of the CRN domain with the participants of this study. Third, it assumes that the gold standard used in the precision and recall measurements covers all objects that should be returned given any of the analyzed changes. One way to disprove the applicability of approach is to attempt to break the first two assumptions by including more end-users in the evaluation, to see if there is more to the domain knowledge currently manifested in the approach framework. While the participating CRN stakeholder is from the highest level of the CRN and should thus be considered as highly knowledgeable about business processes, he may not be privy to every minute detail of the day-to-day activities of the medical coders and others who are involved in managing changes. These associates may be operating with their own set of unwritten "CIARules" in their minds. Furthermore, individual associates may have each their unique rule sets, or each their unique ways of expressing the same rule set. Thus, a gold standard defined by these associates, who do not yet know of our formally defined CIARules, might look entirely different from our gold standard when given the same set of changes to analyze. Their gold standard could contain more objects in each AIS, fewer objects, and different objects, all of which cases would imply different performance measurements. Our own quantitative evaluation shown here served perhaps better as a verification of the prototype implementation—it uncovered a defect that had not been revealed in development testing. One other interesting observation made was that the process of defining the data sets for the analyses was essentially a manual CIA process. It was highly time-consuming even for such small sets, and certainly tedious. If this is any indication of the experience had by the CRN associates in the current situation, we expect they would greatly appreciate tool support.

Clearly, the next step forward would be to repeat the evaluation design with end-users of the CRN, outside of the bubble of the MBE4CR project. The evaluation should then also examine usability requirements of the user interface of the implementation (regardless of whether the CRN chooses to use the prototype created by us or one of their own). One potential usability problem is the poor scalability of a list-based presentation of the EIS like the one our

prototype application uses. If a large number of objects are retrieved from an analysis, clicking and scrolling through each of them in a list could be experienced as inefficient and tedious by the user.

5.1.2 Change Support

An important issue of the approach is that of the limited set of CIARules currently defined (presented in section 3.4.3), particularly for the cancer coding rule class of changes. The currently defined rule set for this class covers only the most basic dependencies and impacts; it is regarded as the least complete set for any change type in the change model. Hence, the rather favorable performance measurements presented in 4.2.6 only apply to the changes currently supported by the CIARules; a proposed change to, say, the MAR in the system set would return a precision and recall of zero. More CIARules exist but have not yet been implemented in the CIA prototype tool. One subset of these as of yet unimplemented CIARules includes rules for MARs that are similar to the ones defined for the VARs. Another set of CIARules may arise from one known relation between cancer coding rules: the *order* in which cancer coding rules are applied. The order relation applies to MARs only. A change in the order could prevent other MARs from executing as the (other) MARs could depend on the values from previous MARs. For the MVRs, the rules themselves should be independent, but the same variables are used in multiple rules. E.g., by removing one validation rule that could stop the aggregation process by identifying invalid data, might thus have unknown consequences in other MVRs. For the approach to be fully operational and effective, more CIARules must be defined, verified and validated. It is unclear how many more dependencies exist between the CRN concepts, or how many more must be implemented before the tool would reach its potential. An informal hypothesis of ours is that the number of dependencies existing in the real world is infinite; further work should aim to disprove this with the help of CRN end-users. In any case it must be up to the users how many CIARules must be in place before the tool is useful to them. The main limitation of this rule-based approach, as has also been experienced and noted by for instance Briand et al. [6], is that defining and maintaining the impact analysis rule set is a resource-intensive process.

5.1.3 Scalability

A potential issue, of which we were not able to examine the severity, is the scalability of the proposed approach when applied in the CRN. As discussed in section 3.5.5, scalability here refers to what degree an implementation of the approach could analyze any type of change within a feasible amount of time, using a feasible amount of primary memory. We showed that the complexity of our CIA algorithm is linear in the number of objects returned by all applied CIARules times the time required by the CIARules to retrieve those objects. Actual time and space requirements are implementation-specific and machine-specific. We were only able to test with relatively small data sets already in primary memory, and could not take into account any time used by queries to the cancer object databases of the CRN. We did however show that the potentially most complex CIARules are those for cancer coding rule changes. Hence, focus should be on measuring the output sizes of these rules if implemented to query for objects dependent on a changed cancer coding rule. Keep also in mind that the changed rule must be applied to each and every one of those objects. Furthermore, there is the question of whether or not to temporarily index, cache, retrieved objects in memory, which would require additional time at indexing but save time in future queries. While the approach appears resource-heavy, it is so in the name of precision, as high precision is required to inspire the user's confidence in any tool implementing it.

5.1.4 Implementation Concerns

In section 3.6 we presented the prototype created for the purpose of evaluating the applicability of the approach in the CRN. We mentioned that the prototype manually implements both the domain model and the change model using plain Java. The alternative to manual model implementation would be to generate code from these models using an appropriate model-based engineering tool. As previously noted, the implication of not using the model-based technique for internal model generation is that any changes to the models would not automatically be reflected in the source code. Thus, the developer would have to implement these changes manually. As domain knowledge changes (see sections 2.2.7 and 3.3.2) must be expected, there could be an advantage in using model-based tools in a real-world implementation. Consider also that the change model may have to evolve accordingly. Happily, the MBF4CR framework (introduced in 2.2.6) may already have the necessary facilities to enable automatic generation of model code. Speaking of features provided by the

MBF4CR, retrieving and applying the appropriate cancer coding rules for re-validating and re-aggregating cancer data in a change analysis may be better performed by the cancer coding rule engine of the MBF4CR than the solution implemented in our approach. Recall that the only reason we did not achieve perfect recall was the defect in this part of our tool.

One implication of changes to the domain model is that they could hypothetically obsolete any CIARules that base themselves on the field that was changed (or removed). Ironically, a CIA tool based on this approach would have to be able analyze itself in such a case, or be manually analyzed by a user. Yet another CIARule set would have to be in place for automatic self-CIA of the tool (and what rule set would analyze *that* rule set?). However, the current CIARule set seems to indicate that dependencies arise from only the most fundamental attributes of the CRN domain concepts, such as dates and object identifiers. These are not likely to change unless the CRN desires to uproot its entire architecture, but awareness of this issue should nevertheless be raised.

One final implementation-specific issue is the circular dependency problem that we presented in section 3.5.4. This hypothetical problem, caused by duplicate objects in the EIS, should be expected to prove its existence when new dependency rules for cancer coding rule changes are being defined. The specifics on how to exclude any duplicates from the EIS has been left to the implementation as there are several methods for this, the most appropriate of which could depend on several factors such as programming language and runtime environment.

5.1.5 Comparison to Related Works

We presented related works from the field of software CIA in section 2.3. The proposed CIA approach framework builds mainly upon two key works from the software CIA literature, the first of which is that of Lehnert et al., who address the CIA of heterogeneous software artifacts. Similarly to this approach, we propose a graph-based representation of the interdependent objects in our application domain, in which each labeled node can represent any kind of object. Lehnert et al. analyzes impacts by first building a dependency graph of all artifacts (e.g., system models and source code artifacts) of the system. Then, the impact propagation is analyzed by recursively applying a set of impact propagation rules. In contrast to their approach, we apply dependency rules that query the object stores for only CRN concept objects we know to be impacted by a given type of change, and then we apply impact rules that determine what the impact of the change is given the specific type of object before

adding them to the graph. Thus, we simultaneously build the dependency graph and determine how the change would impact the objects. This simpler design was enabled by the inherent simplicity of our application domain; only change impacts to the cancer message, cancer case, and cancer coding rules need to be considered by our approach. Furthermore, the object querying is necessary for us to be able to re-assess the validity of every impacted cancer data object following some change. This feature was inspired by the second key CIA work by Embury et al., who investigated the impacts of business rules to persistent data objects. Our approach was evaluated to show its performance in terms of precision and recall, and this technique of querying the dependent objects proved to be near-optimalin terms of precision and recall.

The changes that occur in the CRN were classified formally in order to provide a specification of their impacts and to which other objects of the domain that they ripple. Lehnert et al. advocates the use of a standardized, generic change classification (see 2.3.2), but it was decided to create a custom model for our approach. This is mainly because the changes that occur in the CRN are entirely specific to its domain. Furthermore, the generic change model proposed by Lehnert et al. includes change types that do not apply in the context of the CRN, such as the moving of elements or splitting of elements. For this reason, a domain-specific model appears as more parsimonious than a generic one—it needs not propose more classes than should be necessary. As our intent is to provide a domain-specific classification for use in a domain-specific approach, we do not consider our change model to be in a place to contribute to change taxonomies in the field of general software CIA. However, we do propose the model as an example of how changes can be classified in similar contexts of health information registries. Likewise, even though our approach builds upon works of the general software CIA field, we do not expect its conceptual framework to be applicable the other way around. I.e., we do not claim that our approach can be effectively used in general software system CIA.

6 Conclusion

6.1.1 Summary of the Proposed CIA Approach

This work has addressed the issue of analyzing the impacts of changes that occur in the domain of the CRN. The issue, to our knowledge, is specific to the cancer registry domain, but should also be relevant to other medical information systems that gather and aggregate data from multiple sources over time. The changes include the changes to persistent cancer data (cancer messages and cancer cases), the domain model of the CRN, and the special case of business rules known as cancer coding rules. The cancer coding rules constrain the values of the cancer data objects, and changing these values can alter their validity according to the constraining rule.

The proposed framework, approach and prototype implementation were presented in chapter 3. The framework, inspired by related works presented in section 2.3 from the field of software CIA, can be classified as rule-based and model-based. Our approach uses a graph-based representation of the interdependent objects in our application domain, in which each labeled node can represent any kind of object. We apply dependency rules that query object stores for only CRN concept objects we know to be impacted by a given type of change. Then, we apply impact rules that determine what the impact of the change is given the specific type of object before adding them to the graph. Thus, our algorithm simultaneously builds the dependency graph and determines how the change would impact the objects. The object querying is necessary for us to be able to re-assess the validity of every impacted cancer data object following some change.

The changes that occur in the CRN were classified formally in order to provide a specification of their impacts and to which other objects of the domain that they ripple. We created a custom change model for our application domain. We propose the model as an example of how changes can be classified in similar contexts of health information registries. Even though our approach builds upon works of the general software CIA field, we do not expect its conceptual framework to be applicable in general software system CIA, or that the findings of our evaluation would carry any implications to the field of software CIA. The approach may however have its applications in the field of health information systems—perhaps especially in national cancer registries that implement similar data registering processes as

that of the CRN. Section 2.2 presents a study of the CRN and outlines their registering process. We believe that, if the same or similar concept classes and change classes to those of the CRN can be identified within a domain, this approach is likely to be applicable therein.

In chapter 4, we presented a two-stage evaluation of the approach. The results of this evaluation indicate that the approach accurately simulates the changes and their impacts in the context of the CRN, and that the approach prototype tool produces precise estimations of the impacted object sets for the currently supported change types. We showed that its performance in terms was optimal in terms of precision (1.0 in all changes analyzed) in our own evaluation using the prototype tool. In terms of recall, our prototype performed optimally in ~63% of cases. Based on these results, we posit that the approach can be applied to effectively support the data quality measures in the CRN, although more work on the change impact analysis rule set and the prototype tool would be necessary first.

6.1.2 Further Work

We identify four remaining main challenges following our work: (1) the current CIARule set needs improved change coverage; (2) the prototype tool must be integrated in the CRN or a new tool must be made; (3) our prototype is potentially brittle in the face of changes to the CRN domain model; and (4) the scalability of the approach (and prototype tool) is yet to be determined.

The conceptual framework of the approach itself is regarded as stable, but more dependencies and impacts undoubtedly exist between the CRN domain concepts. For one, the current set of CIARules needs to be further enriched in particular when considering the cancer coding rule class of changes. The involvement of potential CRN end-users, including for example medical coders, in establishing more rules should be a top priority. We recommend that the evaluation design outlined in chapter 4 be repeated with these roles as participants.

Furthermore, there are implementation issues that must be considered if the prototype tool were to be deployed in the CRN. First, the tool must be integrated towards the CRN back-ends, so that actual data can be queried. Also, the tool would need integration to their front-end (known as “GURI”). The question remains as to whether this integration is more feasible than building a new API based on the approach framework from scratch. Second, a real-world implementation should investigate a model-based method for implementing the internal

model (the domain and change models) of the approach. This is crucial due to the ever-evolving nature of the CRN domain.

Finally, a remaining issue is that of the scalability of the approach; we could find no way in which to assess this thoroughly, theoretically or otherwise, using the prototype tool. Thus, stress testing and benchmarking of any future implementation should prove interesting.

In addressing these remaining issues, we believe that our approach can greatly benefit the CRN. In conclusion, relating to our aim stated in section 1.2, we claim that our framework can support highly reliable automated CIA in the CRN once change support has been expanded, but that the user-friendliness and analysis efficiency require further verification.

Appendix

A. List of Abbreviations

ACRS	Automated Cancer Registry
AIS	Actual Impact Set
API	Application Programming Interface
CCVR	Cancer Case Validation Rule
CCR	Cancer Coding Rule
CCRC	Cancer Coding Rule Change
CFVR	Common Field Validation Rule
CiN	Cancer in Norway
CIA	Change Impact Analysis
CIARule	Change Impact Analysis Rule
CRN	Cancer Registry of Norway
CRVaT	Cancer Registry Validation and Aggregation Tool
DC	Data Change
DKC	Domain Knowledge Change
EIS	Estimated Impact Set
EMF	Eclipse Modeling Framework
GUI	Graphical User Interface
KREMT	Kreftregisterets Elektroniske Meldetjeneste

MAR	Message Aggregation Rule
MBE4CR	Model-Based Engineering 4 [for] Cancer Registry
MBF4CR	Model-Based Framework 4 [for] Cancer Registry
MVR	Message Validation Rule
OCL	Object Constraint Language
OMG	Object Management Group
OO	Object-Oriented
SIS	Starting Impact Set
UML	Unified Modeling Language
VAR	Validation Rule
XML	eXtensible Markup Language

B. Complete Set of Cancer Coding Rules Used in the Evaluation

CCR Type	Field Name				OCL Expression or Function Description
	ruleID/Object ID	ruleNumber	diagnosisStart	diagnosisEnd	
MVR	357	V14	01-Jan-1993	31-Dec-2016	Basis in ['36', '60'] implies not (Metastase in['0','5', '8', '9', 'D'])
	358	V14	01-Jan-2017	Infinity	Basis in ['36', '60'] implies not (Metastase in['0','5', '9', 'D'])
	386	V29	01-Jan-1993	31-Dec-2016	(Morfologi in['720039', '730019', '730039', '740009', '740509', '741009']and MessageType != 'O') implies Basis in ['57', '72', '98']
	387	V30	01-Jan-2017	Infinity	(Morfologi->startswith('9990') and MessageType in ['H', 'O']) implies Basis = '98'
	388	V30	01-Jan-1993	31-Dec-2016	Surgery = '00' implies MessageType in ['K', 'R']
	389	V30	01-Jan-1993	31-Dec-2016	Surgery = '00' implies MessageType ='K'
CFVR	360	V16	01-Jan-1993	31-Dec-2016	Basis = '98' implies (Morfologi in['340619', '340709', '340809', '341139', '720039','730019', '740009', '740509', '741009', '730039', '750009'] or Morfologi->startswith('9990'))
	361	V16	01-Jan-2017	Infinity	Basis = '98' implies Morfologi->startswith('9990'))
	329	V19	01-Jan-1993	31-Dec-2016	Radiotherapy in [null, '0', '1', '2', '3', '4', '5', '9']
	400	V19	01-Jan-2017	Infinity	Radiotherapy in [null, '1']
	365	V20	01-Jan-1993	31-Dec-2016	HormoneTherapy in [null,'0', '1', '2', '9']
	366	V20	01-Jan-2017	Infinity	HormoneTherapy in

					[null, '1']
	367	V21	01-Jan-1993	31-Dec-2016	ChemoTherapy in [null, '0', '1', '9']
	368	V21	01-Jan-1993	Infinity	ChemoTherapy in [null, '1']
	373	V25	01-Jan-1993	31-Dec-2016	DS ='5' implies Basis in ['00', '10', '20', '29', '30', '31', '40', '41', '71', '90', '99']
	402	V25	01-Jan-2017	Infinity	DS ='5' implies Basis in ['00', '10', '20', '29', '30', '31', '40']
	374	V27	01-Jan-1993	Infinity	DS = '1' implies (Morfologi->substring(5,1) in ['0', '1', '2'] or Morfologi in ['690099', '699999"])
	384	V27	01-Jan-1993	31-Dec-2016	DS = '4' and MessageType != 'D' implies (Morfologi->substring(5,1) in ['3', '6'] or Morfologi in['340709', '740009', '740509', '908019', '390099', '399999', '690099', '699999'])
	385	V29	01-Jan-2017	Infinity	DS = '4' implies (Morfologi->substring(5,1) = '3' or (Morfologi = '908019' and Topografi->startswith('62')))
MAR		A1			The value of diagnosisDate is set to the earliest eventDate found in all related CancerMessages.

Table A-1: The complete cancer coding rule system subset included in the evaluation stage 2 (chapter 4.2).

References

1. Lee M., Offutt A. J., Alexander R. T. Algorithmic analysis of the impacts of changes to object-oriented software. 34th International Conference on Technology of Object-Oriented Languages and Systems2000.
2. Arnold R. S., Bohner S. A. Impact Analysis - Towards a Framework of Comparison. IEEE Conference on Software Maintenance Montreal, Canada: IEEE; 1993.
3. Tóth G., Hegedűs P., Beszédes Á., Gyimóthy T., Jász J., editors. Comparison of different impact analysis methods and programmer's opinion: an empirical study. PPPJ 2010: 8th International Conference on the Principles and Practice of Programming in Java; 2010 2010-09-15; Vienna, Austria. ACM New York, NY, USA.
4. Lehnert S. A Review of Software Change Impact Analysis2011. Available from: https://www.researchgate.net/publication/267386155_A_review_of_software_change_impact_analysis_A_Review_of_Software_Change_Impact_Analysis.
5. Li B., Sun X., Leung H., Zhang S. A survey of code-based change impact analysis techniques. Software Testing, Verification and Reliability. 2012.
6. Briand L., Labiche Y., O'Sullivan L., editors. Impact Analysis and Change Management of UML Models. International Conference on Software Maintenance; 2003; Amsterdam, The Netherlands. IEEE.
7. Goknil A., Kurtev I., Berg K. v. d. A Rule-Based Change Impact Analysis Approach in Software Architecture for Requirements Changes2016. Available from: https://arxiv.org/ftp/arxiv/papers/1608/1608.02757.pdf.
8. Lehnert S., Farooq Q.-u.-a., Riebisch M. Rule-Based Impact Analysis for Heterogeneous Software Artifacts. 17th European Conference on Software Maintenance and Reengineering; Genova, Italy2013.
9. Embury S. M., Wilmor D., Dang L. Assessing Impacts of Changes to Business Rules through Data Exploration. International Conference on Software Engineering Advances; Tahiti, Tahiti2006.
10. Stevens W. P., Myers G. J., Constantine L. L. Structured design. IBM Systems Journal. 1974;13(2):115-39.
11. Freedman D. P., Weinberg G. M. A Checklist of Potential Side Effects of Maintenance Change. Techniques of Program and System Maintenance1981. p. 93-100.
12. Wang S., Lu H., Yue T., Ali S., Nygård J. Domain Analysis in Cancer Registry of Norway (MBE4CR Project).
13. Wang S., Lu H., Yue T., Ali S., Nygård J. MBF4CR: A Model-Based Framework for Supporting an Automated Cancer Registry System. The 12th European Conference on Modelling Foundations and Applications (ECFMA)2016. p. 191-204.
14. Om Kreftregisteret [Available from: www.kreftregisteret.no/Generelt/Om-Kreftregisteret.
15. Innrapportering [Available from: www.kreftregisteret.no/registrene/innrapportering.
16. UML Class and Object Diagrams Overview - common types of UML structure diagrams [Available from: http://www.uml-diagrams.org/class-diagrams-overview.html.
17. UML class [Available from: http://www.uml-diagrams.org/class.html.
18. Object Constraint Language, Version 2.4 [Available from: http://www.omg.org/spec/OCL/2.4/PDF/.
19. GitHub - dresden-ocl/dresdenocl [Available from: https://github.com/dresden-ocl/dresdenocl.
20. Lindvall M. Evaluating Impact Analysis – A Case Study. Empirical Software Engineering. 1997;2(2):152-8.

21. Lehnert S., Farooq Q.-u.-a., Riebisch M. A Taxonomy of Change Types and its Application in Software Evolution. IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems Novi Sad, Serbia: IEEE; 2012.
22. Bohner S. A., editor Software Change Impacts - An Evolving Perspective. IEEE International Proceedings on Software Maintenance; 2002; Montreal, Canada. IEEE.
23. Eclipse Modeling Project [Available from: http://www.eclipse.org/modeling/emf/.
24. Eclipse Neon [Available from: http://www.eclipse.org/neon/noteworthy/.
25. Weiss M. A. Algorithm Analysis. Data Structures and Algorithm Analysis in Java. Third Edition ed: Pearson Education Limited; 2012. p. 49-52.
26. Manning C. D., Raghavan P., Schütze H. Evaluation in Information Retrieval. An Introduction to Information Retrieval, Online Edition. Cambridge, England: Cambridge University Press; 2009. p. 151 - 6.