Use of Global Consistency Checking for Exploring and Refining Relationships between Distributed Models: A Case Study

Yasaman Talaei Rad and Ramtin Jabbari
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 35 weeks of full time studies.

Contact Information:

Authors:
Yasaman Talaei Rad
E-mail: yasi.tr@gmail.com

Ramin Jabbari
E-mail: jramtin@gmail.com

External advisor:
Dr. Mehrdad Sabetzadeh
Research Scientist, Simula Research Laboratory, Norway
Mobile: +47 40 43 67 99
E-mail: mehrdad@simula.no

University advisor:
Dr. Cigdem Gencel
School of Computing, BTH.
E-mail: cigdem.gencel@bth.se

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
CONTENTS

Abstract ........................................................................................................................................... 5
LIST OF FIGURES .......................................................................................................................... 6
LIST OF TABLES ............................................................................................................................... 7
ACKNOWLEDGMENT ....................................................................................................................... 8
Glossary ................................................................................................................................................ 9

1. INTRODUCTION .......................................................................................................................... 11
   1.1. Purpose Statement ................................................................................................................ 13
   1.2. Aims and Objectives ........................................................................................................... 13
   1.3. Research Questions ............................................................................................................. 14
   1.4. Methodology ....................................................................................................................... 14
   1.5. Structure of Thesis .............................................................................................................. 15

2. BACKGROUND ............................................................................................................................ 16
   2.1. Inconsistency Management .............................................................................................. 16
   2.1.1. Consistency Checking Types ......................................................................................... 16
   2.1.2. Consistency Checking of Multiple Models .................................................................... 17
   2.1.3. Other Consistency Checking Methods ......................................................................... 19
   2.2. Tool Support ...................................................................................................................... 20

3. A NEW METHODOLOGY FOR GCC OF DM ............................................................................. 21
   3.1. Methodology Description .................................................................................................... 21
   3.2. Rule Description .................................................................................................................. 25

4. CASE STUDY .................................................................................................................................. 27
   4.1. Case Study1: Warehouse ..................................................................................................... 28
   4.1.1. Data Collection ............................................................................................................. 28
   4.1.2. Model Designs .............................................................................................................. 28
   4.1.3. Case Study Conduct ...................................................................................................... 32
   4.1.4. Resolution Approach .................................................................................................... 35
   4.1.5. Analysis and Results .................................................................................................... 43
   4.1.5.1. Value Categorization of Inconsistencies .............................................................. 52
   4.1.5.2. False Positive ......................................................................................................... 52
   4.1.5.3. Correct Inconsistencies ......................................................................................... 53
   4.1.5.4. False Negative ........................................................................................................ 53
   4.2. Case Study2: Hospital ........................................................................................................ 54
   4.2.1. Data Collection ............................................................................................................ 54
   4.2.2. Model Designs ............................................................................................................. 54
   4.2.3. Case Study Conduct .................................................................................................... 54
   4.2.4. Analysis and Results ................................................................................................... 55
   4.2.4.1. Value Categorization of Inconsistencies .............................................................. 56
ABSTRACT

Context. Software systems, becoming larger and more complex day-by-day, have resulted in software development processes to become more complex to understand and manage. Many companies have started to adapt distributed software engineering practices that would allow them to work in distributed teams at different organizations and/or geographical locations. For example, model-driven engineering methods are being used in such global software engineering projects. Among the activities in model-based software development, consistency checking is one of the widely known ones. Consistency checking is concerned with consistent models; in particular, having a consistent group of multiple models for a whole system, e.g., multiple models produced by distributed teams.

Objectives. This thesis aims to find out how ‘Global Consistency Checking (GCC)’ can be utilized for exploring inconsistency problems between distributed models; particularly among UML class diagram relationships (in terms of consistency), as well as how GCC can be scaled with large number of models and relationships. Thereby, these inconsistencies are also aimed to incrementally resolve in our approach.

Methods. We made a review in distributed software development domain and model management, in particular, methods of consistency checking between ‘Distributed Models (DM)’. Next, we conducted two case studies in two problem domains in order to apply our ‘consistency checking methodology’. We concurrently constructed and implemented new consistency rules, most of which are gathered from literatures and brainstorming with our coordinators. Generally, the method contains implementing different models of the case studies with a tool support and trying to figure out overlaps, merging models and checking the merged model against the consistency rules, and evaluating the results of GCC. We mainly addressed issues focused on consistency checking of individual models and the mapping between them e.g., pair-wise consistency checking (PCC), which are incapable of fully addressing problems against any consistency rules encountered in distributed environments.

Results. We have identified seven types of inconsistency, which are divided in two groups named ‘Global inconsistency’ and ‘Pair-wise inconsistency’. In the first case study, we have 94 global inconsistencies and 73 pair-wise. In the second one, 14 global and 25 pair-wise inconsistencies are resulted. During ‘Resolution approach’, we followed six steps as a ‘systematic procedure’ for resolving these inconsistencies and constructed new merged model in each iteration. The initial merged model (inconsistent model) as an input for the first step has 1267 elements, and the consistent merged model (the output) from the sixth step has 686 elements. ‘time duration’ and ‘required effort’ for checking consistency against each ‘consistency rule’ were recorded, analyzed and illustrated in Sections 4.1.5 and 4.2.4.

Conclusions. We concluded that GCC enables us to explore the inconsistencies, inclusive of resolving them and therefore, refining the relationships between different models, which are difficult to detect by e.g., a pair-wise method. The most important issues are: The number of model comparisons conducted by PCC, The inability of PCC for identifying some inconsistencies, Model relationships refinement and classification based on PCC approach will not lead to a final consistent DM, whereas, GCC guarantees it. Consistency rules application, inconsistency identification and resolving them could be generalized to any UML class diagram model representing a problem domain within the fields of consistency checking in software engineering.

Keywords: Software development, model-driven engineering, homogeneous models, model management, consistency checking.
LIST OF FIGURES

Figure 1. Multiple Parents ................................................................. 17
Figure 2. Multiple Parents within 4 Model ........................................... 18
Figure 3. Multiple Parents within a 4-Model-schema ................................ 18
Figure 4. GCC Methodology ................................................................ 23
Figure 5. Star Mapping Model ............................................................. 24
Figure 6. One against (N-1) Models-(5models) ...................................... 24
Figure 7. Part1-Order (Class view) ......................................................... 28
Figure 8. Part1-Order(Entitiy relation view) ........................................... 29
Figure 9. Part2- Picking ................................................................... 29
Figure 10. Part3- Sequencing ............................................................... 29
Figure 11. Part4- Loading ................................................................... 30
Figure 12. Part5- Replenishing ............................................................. 30
Figure 13. Whole Model .................................................................... 31
Figure 14. One Model against (n-1) Models- (16 models) ...................... 32
Figure 15. Model V1 in First Steps ...................................................... 33
Figure 16. Picking Request Class Diagram in UML ................................. 34
Figure 17. Picking Request Class Diagram in TreMer* ............................ 34
Figure 18. Inconsistency Report .......................................................... 35
Figure 19. Connector and Matching ..................................................... 36
Figure 20. Connector ....................................................................... 36
Figure 21. Map C2 ........................................................................... 37
Figure 22. Identically-named Attributes between Individual Entities and Relationships .................................................. 38
Figure 23. Mapping V12 .................................................................... 38
Figure 24. Object with Multiple Parents .............................................. 38
Figure 25. Multiple parents, Inconsistency resolving ............................... 39
Figure 26. Parallel Edges Aggregation ................................................ 40
Figure 27. Parallel Edge Mapping ........................................................ 40
Figure 28. Parallel Edges Un-typed ...................................................... 41
Figure 29. Parallel Edges Mapping and Refuting .................................... 42
Figure 30. Unnamed Nodes ................................................................ 46
Figure 31. Identically-named Entities and Relationships ......................... 46
Figure 32. Disconnected Nodes .......................................................... 46
Figure 33. Distinct Attribute Names .................................................... 47
Figure 34. Disjoint Name sets for Attributes and Links ............................ 47
Figure 35. Navigable Nodes ................................................................ 47
Figure 36. Distinct Attribute types and Inherited types ............................ 48
Figure 37. Dangling Edges ................................................................ 48
Figure 38. Type Violation .................................................................. 48
Figure 39. Multiple Inheritance .......................................................... 49
Figure 40. Parallel Edges(strict) ......................................................... 49
Figure 41. Parallel Edges(loose) .......................................................... 50
Figure 42. Cyclic Inheritance ............................................................... 50
Figure 43. Cyclic Composition ............................................................. 50
Figure 44. All Consistency checking Rules Diagrams ............................... 51
Figure 45. Star Mapping .................................................................... 55
Figure 46. Mapping Evlolution1.2 ......................................................... 60
LIST OF TABLES

Table 1. Glossary .................................................................................................................................................. 10
Table 2. Types of Consistencies .......................................................................................................................... 16
Table 3. Consistency checking Strategies ........................................................................................................... 16
Table 4. Inconsistency Management Activities ................................................................................................. 18
Table 5. Rule Implementation ............................................................................................................................ 25
Table 6. UML Element count of Different Parts of a Model ................................................................................ 31
Table 7. Relation Description .................................................................................................................................. 33
Table 8. Inconsistencies in Each Stage ................................................................................................................ 44
Table 9. Warehouse Inconsistencies .................................................................................................................... 45
Table 10. Element Counts of 16 Models ................................................................................................................ 45
Table 11. Effort level for Each Consistency Resolving Stage .............................................................................. 45
Table 12. Value Categorization of Inconsistencies for Warehouse ....................................................................... 52
Table 13. False Positive Inconsistencies for Warehouse ..................................................................................... 53
Table 14. Correct groups of Inconsistency in Warehouse ..................................................................................... 53
Table 15. False Negative ........................................................................................................................................ 54
Table 16. Hospital Inconsistencies ....................................................................................................................... 56
Table 17. Value Categorization of Inconsistencies for Hospital .......................................................................... 56
Table 18. False Positive Inconsistencies for Hospital ........................................................................................ 56
Table 19. Correct groups of Inconsistency in Hospital .......................................................................................... 57
Table 20. Entity-Relationship Count (First and last step) .................................................................................. 59
Table 21. Element Correspondence of the Connector .......................................................................................... 59
Table 22. Mapping Evolution of Models ............................................................................................................ 60
ACKNOWLEDGMENT

We would like to thank our supervisors Dr. Cigdem Gencel and Dr. Mehrdad Sabetzadeh for their supports, guidance and insightful ideas throughout this work. This thesis would not have been possible without their valuable time and input. We are thankful for the constructive meetings and discussions that we had, which were a great inspiration and helped us to focus on the subject and most important issues.

We express our gratitude to our parents for their sincere and constant support during all stages of our life. We never forget their encouragement and support during all days of our life.

Yasaman, Ramtin
**Glossary**

Table 1 describes the used terms in this thesis.

<table>
<thead>
<tr>
<th>Conceptual Modeling</th>
<th>“A mental model captures ideas in a problem domain, while a conceptual model represents ‘concepts’ (entities) and relationships between them. A Conceptual model in the field of computer science is also known as a domain model. The aim of a conceptual model is to express the meaning of terms and concepts used by domain experts to discuss the problem, and to find the correct relationships between different concepts. The conceptual model attempts to clarify the meaning of various, usually ambiguous terms, and ensure that problems with different interpretations of the terms and concepts cannot occur. A conceptual model can be described using various notations, such as UML or OMT for object modeling. In the UML notation, the conceptual model is often described with a class diagram in which classes represent concepts; associations represent relationships between concepts and role types of an association represent role types taken by instances of the modeled concepts in various situations.” [15]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Development</td>
<td>Distributed development is the allocation of work to distributed teams that are working on the same project (parallel or other strategies) and integrate all parts done by the distributed teams, as a whole project, at the end of endeavor. It is joint-performance. [16, 17]</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>“Models may be built by teams with different responsibilities and goals. Hence, models may have conflicting purposes or disagreements over the choice of ‘Terminology’, ‘Design’, and ‘Usage’.” [1]</td>
</tr>
<tr>
<td>Model-driven engineering</td>
<td>“Model-driven engineering (MDE) is a software development methodology which focuses on creating and exploiting domain models (that is, abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts. The MDE approach is meant to increase productivity by maximizing compatibility between systems (via reuse of standardized models), simplifying the process of design (via models of recurring design patterns in the application domain), and promoting communication between individuals and teams working on the system (via a standardization of the terminology and the best practices used in the application domain). A modeling paradigm for MDE is considered effective, if its models make sense from the point of view of a user that is familiar with the domain, and if they can serve as a basis for implementing systems.” [18]</td>
</tr>
<tr>
<td>Software Development Process</td>
<td>The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. [19]</td>
</tr>
<tr>
<td>TreMer*</td>
<td>TreMer* is a tool for editing and visualizing models, and EPS Graphics2D (<a href="http://www.jibble.org/epsgraphics/">http://www.jibble.org/epsgraphics/</a>) for exporting models to PostScript vector graphics. It is written in Java that has roughly 15K lines of code, of which 8.5K implement the user interface, 5.5K implement the tool’s core functions (model merging, traceability, and serialization), and 1K implement the interface for interacting with CrocoPat. The tool uses Jgraph (<a href="http://www.jgraph.com/">http://www.jgraph.com/</a>) TreMer* was publicly released in May 2007. The tool is freely available at <a href="http://www.cs.toronto.edu/_mehrdad/tremer">http://www.cs.toronto.edu/_mehrdad/tremer</a>. [1]</td>
</tr>
<tr>
<td>UML</td>
<td>“The Unified Modeling Language (UML) is a general-purpose modeling language for visualizing, specifying, constructing and documenting the</td>
</tr>
</tbody>
</table>
artifacts of software-intensive systems. The UML was originally derived from the object modeling languages of three leading object-oriented methods: Booch, Object Modeling Technique (OMT) and Object-Oriented Software Engineering (OOSE). It was first added to the list of OMG adopted technologies in 1997, and has since become the industry standard for modeling software-intensive systems.” [20]  

Table 1. Glossary
1. INTRODUCTION

Software systems are becoming larger and more complex day-by-day. With the software and its development process growing in scale, software development also becomes more complex to understand and manage. Market competition forces software producer companies to develop their software products, which would be cheaper, faster and have better quality. As a result, many companies are adapting distributed software development practices that would allow them to work in distributed teams at different organizations and/or geographical locations e.g., global software engineering. It could help software developers to handle the development processes in terms of understandability, time, cost and quality. Model-based software development i.e., modeling, concerned as a collaborative effort that could be distributed between distributed development teams, hence, it seems that model management as a whole, will become a major issue in the coming future [53].

For this purpose, we found model management activities interesting enough to work on this domain. On the other hand, consistency management and, in particular, consistency checking are the widely known set of activities in model management. Moreover, GCC as a new method for consistency checking by the use of ‘Merge’ and ‘Match’ operators guided us to turn towards conducting needed study on consistency management through GCC methodology.

The work in this thesis concentrates on the challenges posed by distributed software engineering in a model-based software development context. In particular, the requirements analysis phase in such a development results in having multiple distinct but overlapping conceptual models, used to capture the needs and goals of different stakeholders. It is obvious that, in Model Driven Engineering (MDE), [2] models play a vital role in all dimensions of such a development. Models are built to enable analysts to better understand the problem domain. Problem domain usually is described by stakeholders’ goals and requirements.

According to existing literatures, there are four basic distribution dimensions in software development as geographical, organizational, temporal distribution, and shared (or distributed) work [54]. In large software projects, ‘modeling’ is considered as a collaborative effort that could be distributed between distributed development teams. In other words, modeling could be considered as a shared work between modelers, analysts, developers, etc. Therefore, this thesis has not been influenced by different geographical, temporal and other related factors.

Teams usually work on inter-related models representing different sources of information, or information of different development concerns. “Model management is concerned with describing the relationships between these models, and providing systematic ways to analyze and manipulate the models and their relationships.” [21].

‘Consistency checking’ is the most important activity in ‘model management’, which aims to ensure consistency between a set of models of a whole system. The models may have different criteria for the notational constraints of the modeling language and quality constraints, for increasing the understandability, maintenance and evolution [3].

Consistency checking has many complexities and different concepts as a research topic. Since 1980’s, studying on consistency checking has indicated that consistency has been considered as a main challenge in software engineering. During recent decades, some techniques have been introduced for
different types of software development artifacts such as: logical specifications, conceptual graphs, state diagrams, XML documents, UML models, natural language documents and so on[1].

Our study focuses on UML models, in particular class diagrams for checking consistency in order to improve relationships between DM. Class diagrams are the main basis of object-oriented design and analysis [22]. Class diagrams show the system in classes and their inter-relationships, operations and attributes [23].

The initial explanations of requirements, which are elicited from stakeholders, are usually informal and interpret in a different way cause to misunderstanding points [24]. Therefore, ensuring the accurate requirements specification, as much as possible, is one of the main concerns for requirements engineers [25].

Consistency, completeness and correctness are usually mentioned as controversial quality attributes in literature, particularly, in initial phases of software development by requirements engineers to conduct accurate requirements [25]. The main reason for that is regarded to the complexity growth of software projects, technological development and client requests, all of which cause to have possibly vague requirements.

A sample for the case can be the expectation of delivering the project with numbers of requirements elicited from different stakeholders that is developed by e.g., distributed team(s), and it should be delivered on time with the desired quality. As a consequence, providing the software deliverables in each development-phase should be handled by the distributed teams in different locations and it will cause to inconsistency problems between the outputs built by distributed teams.

In requirement phase, one of the main problems is consistency detection between requirements, which are elicited from different stakeholders. Distributed development, different stakeholders and different requirements result in having inconsistent and conflict interpretations between requirements [1].

Consistency checking will be accomplished on individual models, pairs of models or even collections with multiple models and multiple relationships. Conceptual modeling is usually a collaborative effort between teams of people, which can be in the same or distributed organizations. These teams construct individual models, most of which overlap each other. These overlapping models indicate different viewpoints on the overall system [3]. In design phase, for modeling elicited requirements, there are some architecture patterns such as Object Management Group (OMG) that introduce Model-Driven Architecture (MDA) to e.g., provide a standard for automatic translation of Unified Model Language (UML) models, to e.g., object oriented code [26]. The main aim of automatic translation of UML models to code is to minimize the number of errors and develop more reliable code in comparison with manual translation. Hence, having the more consistent UML models lead to result in the more reliable and conformed code. The usefulness of checking consistency and consistency validation between UML models is that it is harder to make changes in source code than in UML models. Consistent models also cause to minimize the cost over development process [27]. Therefore, ensuring the consistency between models plays a key role in a successful model-driven software development, in particular, when a software system is developed by distributed teams in distributed development approach.

In literature, there are some definitions of model consistency, which are interpreted in various contexts, and sometimes it seems, some of them are ambiguous and contradictory in comparison with each other [30]. In order to handle these ambiguities, we reference to [29] to define the term “consistency”, which is “a state, in which two or more elements, which exist in different models of the
same system, have a satisfactory joint description and/or the same concept”. In this definition, a group of activities for diagnosing and handling inconsistencies is named “consistency management”.

Many root-causes are introduced for inconsistency problems; In [28], ‘multiple views’ (different sources of information) and ‘distributed development’ are notably mentioned. We have also studied these two root-causes in this thesis, in order to check consistency between DM.

‘Multiple views’, most of the times, is the nature of a system that has different stakeholders with different concerns. These views sometimes overlap each other due to some ambiguities and even lead to conflicts between them, all of which are making the models inconsistent.

‘Multiple views’ sometimes come from development process. A system development process may have multiple phases, iterations or increments, each of which can produce inter-mediate products or evolve the system through phases.

‘Distributed development’ also causes to inconsistencies due to distributed development teams (local or global) with different views and different interpretations of requirements [28].

Therefore, consistency checking is a vital activity in distributed development including different stakeholders with different concerns on a system [1].

1.1. Purpose Statement

The purpose of this thesis is to explore the factors affecting consistency of DM using GCC. This involves the study of how GCC can be implied for handling consistency problems as the main activity in model management. For evaluating the results practically, two case studies were conducted in two different problem domains.

Model management’s perspective and the conceptual models are elicited by distributed development teams. They were used to develop an understanding of different factors and different stakeholders’ viewpoints. Moreover, the possibility to propose a method for GCC in comparison with pair-wise checking has been explored.

1.2. Aims and Objectives

The main aim of this thesis is to find out how GCC can be utilized for exploring and refining the UML class diagram relationships (in terms of consistency) between DM.

Our aim is to provide a more comprehensive treatment of inconsistency resolution and pre-defined procedures for addressing different types of inconsistency. In fact, the goal is to provide more definitive answers to whether or not GCC is feasible and useful.

For this purpose the following objectives are fulfilled;

- Identify different factors affecting consistency of a system in software engineering context.
- Evaluate and validate the usefulness/effectiveness of GCC in practice, for improving the relationship between DM.
- Investigate if GCC can be applied practically in comparison with pair-wise as suggested in literature for utilization in distributed software development, keeping in view both stakeholders’ and distributed development teams’ perspectives.
- Merge DM and imply the consistency rules on the whole system of the models, instead of making comparison in pairs of models.
- Use the traceable information, which we have merged models back to the origin of each model and inspect why an inconsistency has been raised and find list of issues.
- Find out how GCC can be scaled with large number of models and relationships.
- Find out the possibility of incrementally resolving inconsistencies of DM.
- Find out how elaborations of resolutions will affect all mappings and models’ relationships.

1.3. Research Questions

RQ1. Does GCC enable finding inconsistencies, which are hard to detect or not detectable at all by using PCC?
RQ2. Does GCC scale when we have a large number of models and mappings?
RQ3. Is GCC feasible to incrementally resolve (elaborate) inconsistencies between DM?
RQ4. How does the elaboration affect the mappings?

1.4. Methodology

In order to answer the research questions, we designed our research as below:

We focused on conducting a preliminary study to evaluate GCC, as a recent idea for consistency checking by the use of merge and match operators in model management domain. However, we started by reviewing [1] as the main literature. Next, we continued with the references of it. We reviewed all those references by reading their abstract, objectives and conclusion sections. We selected from the references, those ones were related to our study, particularly, consistency checking of UML models. Then, we read the whole contents of selected references to find more relevant ones, which are [2, 3, 4, 5, 7, 8, 21, 29, 33, 34, 35, 36, 37, 39, 40, 41 and 51].

Next, we checked references of new selected papers, based on their titles. We reviewed their abstract, objectives and conclusion section as well. Then, we finalized this round of reviewing existing literatures. Meanwhile, we searched in some academic databases to find other relevant papers in this domain for better understanding the conventional methods of consistency checking and tool support lacks, specifically for UML class diagram models, which are [6, 11, 13, 14, 22, 23, 24, 25, 27, 30, 31, 32, 38, 42, 43, 44, 47 and 48]. It should be mentioned that, [28] in particular, (as a systematic review of UML model consistency management, conducted by University of Murcia in Spain) had an essential role in the outputs of the review. We examined the cited literature in this review, in particular, in search of consistency checking rules. The outputs are ‘the enhancement of the methodology’ for consistency checking, which came from [1] and discussed in Section 3, and define ‘new consistency checking rules’, which are implemented in the tool support (Section 2.2) [1, 6, 13, 14, 26]. It should be mentioned, the review continued over the thesis, if needed.

The theory for consistency checking via merge and match and the methodology of use in [1] needed preliminary studies. So, it leaded us to conduct an exploratory study as our contribution in the thesis to validate the methodology and new implemented rules, and try to answer whether GCC is feasible and useful for UML class diagram models or not.

In this thesis, we conducted two case studies by collecting data from two different problem domains in order to ensure about the results regardless of the nature of problem domains. Also, the strategies (i.e., the initial system of models) which were used in the case studies were different as well. The data were
in the form of raw models (See Appendix B), which were developed by senior students (mostly graduated) [12], as a project of object-oriented analysis and design. These models were in class diagram notations and elicited from different stakeholders (See Section 4.1.2 and Appendix B).

Therefore, this thesis covers various aspects of model management, as follows;

- How to identify and refine the relationships between independently-developed models.
- How to combine models with respect to known or hypothesized relationships between them.
- How to ensure consistency between models originating from different sources.
- How to propagate changes made to one model to other models related to it.

And here, the way of answering the research questions are discussed;

For answering RQ1, both case studies have been selected including multiple models, which are elicited from different source of information. GCC methodology has been applied to the models and the rate of inconsistencies has been discussed (See Section 4).

For RQ2, scalability is defined in this thesis by five factors;

1. Models size (number of elements) (See Table10),
2. Duration time of GCC against each consistency rule (See Figure 30 to Figure 44),
3. Rate of global and pair-wise inconsistencies (See Table 9 and Table 16),
4. Number of inconsistencies in each stage of inconsistency resolution (See Table 8),
5. Estimate required effort for resolving each group of inconsistencies (See Table 11).

In order to answer RQ3 and RQ4, all inconsistencies which are obtained from the first case study execution (See Section 4.1), have been resolved by the employment of related procedure in six stages of inconsistency resolution (See Section 4.1.4, Table 11 and Figure 30 to Figure 44). During six stages, it is indicated that how the mappings are affected by ‘Elaboration phase’ (See Section 4.1.5). More details about answering RQ4 are available in Section 5.

1.5. Structure of Thesis

This section indicates the structure of the thesis and the contribution of the chapters for answering the RQs.

Chapter 2 (BACKGROUND) describes previous studies on ‘Inconsistency management’ (Section 2.1), including types of consistency checking (Section 2.1.1) and ‘Consistency checking of multiple models’ (Section 2.1.2). Also, some consistency checking methods are described in (Section 2.1.3) to explain and understand the reasons for GCC. ‘Tool support’ (Section 2.2) introduces and describes some existing consistency checking tools and in particular, TreMer™, which is used in our work to implement GCC. The contribution of this chapter is to provide preliminary information for the thesis’ context.

Chapter 3 (A New Methodology for GCC of DM) completely defines our GCC methodology, which is used in Chapter 4 through conducting two case studies. Chapter 3 also introduces new consistency checking rules that were identified from literatures in order to be implemented in TreMer™ (Section 3.2). This chapter contributes to answer research questions (1), (3) and (4).

The execution of the case studies is presented in Chapter 4. It includes ‘Data collection’ and ‘Analysis and results’. This chapter also indicates conceptual models, which are extracted from different
stakeholders individually, as raw inputs for our study (Section 4.1.2 and 4.2.2). Chapter 4 provides a trend of our GCC methodology (Section 4.1.4. Resolution approach) in order to resolve inconsistencies, which we diagnosed. This chapter contributes to answer research questions (1), (2), (3) and (4).

Section 4.1.5 and 4.2.4 represent the contribution of the thesis for the results and analysis of both selected case studies, which mainly provide the feasibility of detection and resolution in rates and time durations.

In Chapter 5 (Discussion), the research problem and the contribution of the thesis are discussed based on the RQs and in Chapter 6, the threats to the validity are also discussed. Chapter 7 concludes the thesis and Chapter 8 presents directions for future work.

2. BACKGROUND

This chapter summarizes the background work on consistency checking. In Section 2.1, we provide a discussion on inconsistency management including types of consistency checking, consistency checking of multiple models, and other consistency checking methods. In Section 2.2, we discuss about some existing tools for checking consistency.

2.1. Inconsistency Management

2.1.1. Consistency Checking Types

There are five main types of consistency defined by Mens et al [31 and 11] (See Table 2).

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical consistency (inter-model)</td>
<td>Consistency is validated at different levels of abstraction between different diagrams. Syntactic and semantic consistencies are also included in it.</td>
</tr>
<tr>
<td>Horizontal consistency (intra-model)</td>
<td>Consistency is validated at a same level of abstraction between different diagrams.</td>
</tr>
<tr>
<td>Evolution consistency</td>
<td>Consistency is validated between different versions of a same UML diagram.</td>
</tr>
<tr>
<td>Semantic consistency</td>
<td>Consistency is validated for UML diagrams semantic meanings defined by UML meta-model.</td>
</tr>
<tr>
<td>Syntactic consistency</td>
<td>Consistency is validated for UML diagrams specifications in UML meta model.</td>
</tr>
</tbody>
</table>

Table 2. Types of Consistencies

Consistency Strategy: It identifies the strategy used to validate consistency between UML diagrams. There are three types of strategy available in literature as consistency by analysis (based on an algorithm), consistency by monitoring (based on rules), and consistency by construction (generates one artifact from another) [32].

For consistency checking, there are three main strategies available in literature (See Table 3).

<table>
<thead>
<tr>
<th>Consistency Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>consistency by Analysis</td>
<td>Based on an algorithm</td>
</tr>
<tr>
<td>consistency by Monitoring</td>
<td>Based on rules</td>
</tr>
<tr>
<td>consistency by Construction</td>
<td>Generates one artifact from another</td>
</tr>
</tbody>
</table>

Table 3. Consistency checking Strategies

In our thesis, as we check the consistency against ‘consistency rules’, our strategy comes from monitoring approach, according to Table 3, through ‘consistency by monitoring’, each new element of a model is checked against existing consistency rules. For instance, there is a tool that provides monitoring facility to check model’s elements against the consistency rules, which have been
implemented in the tool’s repository. It should be mentioned that, the model remains constantly consistent by the use of monitoring approach for consistency checking [32].

2.1.2. Consistency Checking of Multiple Models

There are two ways for consistency checking of interrelated models:

1. **PCC**, which implies the rules on the pair-wise relation of the models. The rules are defined to ensure the well-formed state and compatibility of the relation between the pairs of models.
2. **GCC**, which first combines the different models through merge operator, with respect to their relationships, and then the merged model will be checked against consistency rules.

Figure 1 shows two different models M1 and M2. The example comes from the first case study. In both models Back-Fascia and Rear-Fascia are the same. But according to the multiple inheritance characteristics, the relation R is inconsistent, because it equated two different concepts without equating the parents as well.

![Figure 1. Multiple Parents](image)

Figure 1. Multiple Parents

The example implies for the inconsistency between two models, so in case of having different models, more comparisons are needed and consistency checking would be more complicated.

According to Figure 2, Picker entity in model V1 has different parents as Team in V1, Employee Member in V3, Worker in V16 and Employee in V8, therefore, the inconsistency Multiple parents is resulted. So, the relations R1, R2 and R3 are inconsistent because they equate three different concepts without equating their three parents in the whole model.
According to [29, 33], we define inconsistency management activities in Table 4, as follows:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consistency checking</strong></td>
<td>“Consistency checking between [models] is a vehicle for integrating these [models]. It is the activity in which two or more [models] compare knowledge and ascertain whether or not the relationships that supposedly hold between them do indeed hold” [51].</td>
</tr>
<tr>
<td><strong>Inconsistency classification</strong></td>
<td>To identify the types of inconsistencies according to 1. Causes of inconsistencies, or 2. Predefined types defined by developers.</td>
</tr>
<tr>
<td><strong>Inconsistency handling</strong></td>
<td>To make an appropriate decision against inconsistencies, e.g., circumvent, resolve, tolerate, or ignore it. (Usually it depends on the results of inconsistency measurement.)</td>
</tr>
<tr>
<td><strong>Inconsistency measurement</strong></td>
<td>To calculate the impact of inconsistencies over development process. To prioritize the inconsistencies based on their impact.</td>
</tr>
</tbody>
</table>

Table 4. Inconsistency Management Activities

Our thesis focuses only on “consistency checking”, in particular, checking consistency between UML class diagram models (inter-consistency checking) is the main focus. Therefore, our background section relevant to this domain, governs consistency concepts and its current activities, in particular,
those concepts that are defined for distributed development. More details for inconsistency management are available in [29, 33].

2.1.3. Other Consistency Checking Methods

As our study is on using GCC for improving the relationships between DM, we also describe existing consistency checking methods between models (e.g., UML models) in this section. The majority of these techniques and methods focus on inter and intra level consistency checking between models. Most techniques provide consistency rules as a strategy for consistency checking named monitoring strategy [11]. The studies on consistency checking are as follows;

- **Consistency checking between heterogeneous UML diagrams**
  In [34], a method has been presented for checking consistency between heterogeneous UML diagrams. The well-formed state constraints and using (structural) conformance, as the underlying notion of consistency, has been focused. Consistency rules have been implemented by an imperative programming language. In order to be able to keep track of the elements, each rule’s behavior has been profiled during the evaluation. So, the resulting information is used for correlating between elements, their consistency rules, and inconsistencies. Based on the “resultant information”, the approach can decide about reevaluating the rules and thereby displaying inconsistencies.

- **Consistency checking for multiple view software architectures**
  The context of the consistency management develops a graph-based language [35] to describe software architecture with multiple viewpoints. Also, within this context a framework for checking the consistency intra- and inter- related models are provided. As mentioned in Table 2, the consistency within the model and at the same level of abstraction is *intra-model* consistency and the consistency within different models and different abstraction levels is *inter-model* consistency. The approach includes both logical consistency and conformance. In better words, multiplicity constraints are translated to system of linear inequalities and with the usage of satisfy-ability checking, it ensures about the solution of the system. Also a simple constraint language based on first order logic has been provided by this approach and it describes the process-level constraints over viewpoints and the relations. The conformance is used as the notion for verifying the constraints. By this approach the need for viewpoints combination has been recognized which is exactly before global constraints. But no specifications for this combination have been done by this approach.

- **Merging incomplete and inconsistent state machines**
  The approach is to verify the global consistency of state machines. The behavioral conformance (conformity between entities and their relationships agreements and similarities, e.g., aggregation based on new results) is the underlying notion of consistency. At first, the sources of state machines are merged and then the merge will be checked for the properties of interest.
  The main characteristic of the consistency checking component in this approach is the usage of multiple-value semantics for properties evaluation. This aspect enables the toleration of inconsistencies and provides a solution to classify inconsistencies according to their impacts. So the classification verifies the impact of the inconsistencies and how they can be treated [39].
- **Formal framework for consistency checking**
  This approach has proposed a formal framework for consistency checking of different viewpoints in some languages, like LOTOS (Language of Temporal Ordering Specifications) [40] and Z [36]. The logical consistency checking has been used in this method and the consistency of a collection of viewpoints is resulted if all implemented viewpoints are non-empty. Within this approach, different strategies for PCC have been described. The approach also show that by using iterative merging viewpoints with applying logical unification and binary consistency checking, the global consistency properties can be checked [37].

- **Xlinkit for checking consistency between XML documents**
  It constructs an end-to-end framework in order to check consistency between distributed XML documents. The framework is named Xlinkit. Xlinkit has a document management mechanism based on 1st order logic to develop and define consistency rules. It has also an engine to check documents according to consistency rules and report inconsistencies. Also Xlinkit uses conformance as the notion of consistency. The rules do not directly specify, from where the elements specified in the rules should be retrieved. Hence, the rules are applicable to various sets of documents. Instead of using Boolean semantics for the consistency rules, extended semantics are provided by Xlinkit in terms of hyperlinks. By this aspect, the consistent elements are linked if a consistency rule holds, otherwise inconsistent elements are linked. The base of this linkage refers to the diagnostics that includes only the elements which directly contributed to an inconsistency.

  It should be mentioned that, in this method, ‘document’ is considered as documents contain structured or semi-structured data, such as XML, databases, source code, etc. for more information, see [38].

- **Conformance checking**
  The approach [41] studies the problem for consistency constraints imposed by the UML meta-model (Unified Modeling Language, 2003). The constraints include both individual UML diagrams and pairs of (heterogeneous) UML diagrams. These two consistency checking methods within this approach are as follows:
  - Using a general-purpose theorem proving
  - Imperative object-oriented programming language

  The whole study emphasis that model checking (conformance checking) can verify most meta-model constraints and there are certain constraints which involve checking logical consistency for their verification.

2.2. Tool Support

TreMer+ was used as a tool support in our study. It is presented by Mehrdad Sabetzadeh -who is our shadow supervisor from Simula- and his colleagues to support consistency checking between DM, more details for TreMer+ is published and available in [2]. As it is mentioned in the peer-reviewed paper [2], TreMer+ has an automated matcher and merger for building the initial system of models (See Section 3.1, Figure 4, Phase 2), that is not provided by other existing tools.

Another reason for choosing TreMer+ is that, by the use of this tool and helps of the owner, it was easily possible for us to implement our new consistency rules, which are defined during our studies, as mentioned in Section 3.2.
We briefly study some other tools in this domain for becoming more familiar with the concept behind other existing tools and studying differences. It should be mentioned that, other existing tools have not possibility for checking consistency through merge and match operators, which is described in our study.

In [42], UML/Analyzer is introduced for checking consistency based on the transformation of the model. In other words, by UML/Analyzer, it is possible to check consistency from different forms of the same model, i.e., between class diagram and sequence diagram of the same model. In fact, it checks consistency vertically for individual models at different levels of abstraction. More details are available in [42, 43].

In [44], Behavioral Consistency Checker (BCC) is introduced for checking behavioral consistency on UML models, based on translation of a UML model to ‘Instantiable Petri Nets (IPN)’. BCC was developed during ModelPlex, as the European integrated project. More details are available in [44], [45] and [46].

In [47], Charmy is introduced as a tool for checking consistency of architectural models to provide the consistent models for the software architects. By Charmy, it is possible to draw architectural state diagrams and scenarios through using related editor components. More details are available in [47] and [48].

In [50], DOPLER is introduced as an incremental consistency checker in order to detect and provide information of inconsistency problems. The main reason of DOPLER’s method is to improve the overall performance and scalability of consistency checking activity. More details including how a modeler can work with DOPLER are available in [50].

As mentioned before, other existing tools (some of which are mentioned briefly here) do not work on DM at the same level of abstraction in order to apply e.g., GCC approach. The focus of the majority of these existing tools is on ensuring consistency of individual models or between different forms of the same model e.g., consistency between a class diagram and the same related state machine, etc.

3. A New Methodology for GCC of DM

In this chapter, we introduce our methodology for GCC of DM, which is the enhancement of the previous methodology used in [1]. Also, consistency checking rules, which are used in this methodology, are defined in Section 3.2.

3.1. Methodology Description

It is possible to use GCC for modeling problem domains in e.g., object oriented analysis (OOA). In case of having one or multiple distributed problem domain models, GCC methodology can provide a whole consistent model for a system. The methodology with the usage of match, merge and consistency checking operators, can convert all different inconsistent models to the whole consistent one.

Figure 4 indicated the methodology that we developed for GCC, as an adaptation of previous one in [1]. The brief description of the previous methodology is as follows;
The methodology starts by defining an initial system of models and then computes merge of the system. The results can be as a subject for ‘Manual inspection’ or ‘Automatic consistency checking’ concurrently. These analyses triggers around ‘Elaboration’ as;

- Refining the existing models and relationships
- Defining additional models and relationships
- Revising the existing system of models
- Define a new system of models

Then, it may initiate a new iteration by re-computing the merge and following the subsequent activities. More details are available in [1].

Our GCC methodology, provided in this thesis, mostly uses two operators of model management, which are defined in [1, 21], named Merge and Match (Mapping), provided by the automated matcher (TreMer+), for building the initial ‘system of models’.

The methodology provides mappings between different models. ‘Mapping models’ can be an arbitrary range between none to all of the models. Next phase is to provide a global view through merging all the models in order to have a whole model (the merged models) of the system. Then, consistency of the ‘merged models’ will be checked against ‘consistency checking rules’. As a result, all inconsistencies are diagnosed and then elaborated.

It should be mentioned that, two loops will occur within the methodology; the first one is between ‘Elaboration phase’ and ‘Computing merge’. In this loop, the whole model (elaborated version of the merged models) will be checked again to ensure not having any error. To provide the final consistent model, all relations and entities will be checked again in ‘Manual inspection’. In case of having any undetected inconsistency, the second loop between ‘Manual inspection phase’ and ‘Computing merge’ will occur. Finally, the output is the consistent model (the merged models).

Therefore, we have provided the enhancement of the previous methodology to have a more comprehensive treatment of inconsistency resolution. Pre-defined procedures for addressing different types of inconsistency in this thesis are as follows;

- It uses a more extensive ‘consistency checking rule’ set (twice the number of rules in [1])
- Checking the consistency of each ‘individual model’ before merging them is added, as the first phase.
- ‘Analysis phase’ (in [1]) is divided into two phases, which are ‘Check global consistency’ and ‘Manual inspection’.
  The reason for that is to avoid conflicting or overlapping ‘auto detected inconsistencies’ with ‘manual detected inconsistencies’.
- In this methodology, ‘Elaboration phase’ does not include ‘Define new system of models’.
  The reason for that is to study the results of ‘the initial system of models’, as our aim in this thesis as the initial system of models plays a key role in our approach for GCC.

In fact, these changes have been done to provide more definitive answers to whether GCC is feasible and useful. Figure 4 shows all phases in our methodology explained in following paragraphs.
1. **Individual models consistency checking**: Check ‘each individual model’ to ensure to be consistent.

2. **Initial system of models**: Define initial system of the models. To define the system of models, there are many strategies for connecting the models which can be in an arbitrary range between none to all of them. Two notable strategies are ‘Star’ and ‘1 against (N-1) models’. Making connection between models means to map each model to another one.

   **Star**: Each pair-model is connected via one mapping. In this strategy, every model is mapped to all other ones. The number of mapping between $N$ models is $N \times (N - 1)$ in ‘Star’ strategy. In Figure 5, five models ($N = 5$) are mapped to each other by ‘Star’ strategy. Therefore, we have $5 \times (5 - 1)$ ‘mapping relations’ between the models.
One against (N-1) models: As it is illustrated in Figure 6, in this strategy, one of the models is chosen, named M1 and only M1 is mapped to the other models. Hence, there is not any mapping between the other ones.

In our methodology, the initial mappings are based on ‘Name matching’, which is provided by n-gram algorithm with n=2 [49].

3.  **Computing merge:** Merge the models in order to provide a global view.

4.  **Checking consistency:** Check consistency of the merged models according to the consistency rules, which are introduced in Table 5. In case of finding inconsistencies automatically, the next phase is *Elaboration*. Otherwise, it is ‘Manual inspection’.

5.  **Elaboration:** During this phase, the models and their relationships may be evolved, new models and relationship could be defined and existing system of models may be revised. There are different procedures for resolving the inconsistencies. The procedures differ based on the type of inconsistencies. All the procedures are discussed in ‘Resolution Approach’ (See Section 4.1.4). Then the process goes back to ‘Computing merge’ phase. The loop between ‘Elaboration’ and ‘Computing merge’ continues till no automatic error can be found.

6.  **Manual inspection:** Check the system manually for finding inconsistency problems, such as redundant mapped entities, unmapped relations due to e.g., naming problem (synonyms), etc. In case of having inconsistency problems, go back to ‘Elaboration’.
If the merged model has not any inconsistency problem, global consistency will be achieved.

3.2. Rule Description

In this section, we describe the consistency checking rules of our study. Some rules have been
identified from the literatures, as a more completion ‘rule set’ for our GCC methodology. It should be
mentioned that, in case of analyzing the UML meta-model and restricting the results to simple class
diagrams, the same rules would be found from the existing literature. This thesis focused on checking
consistency of UML class diagram, so our selection criteria for the rules refer to only ‘classes’, ‘class

The consistency checking rules are implemented in TreMer’, as the tool support. Some of them had
been implemented before, and new ones implemented by Simula development team. It should be
mentioned that, one of the contributions in this thesis is to define these new consistency checking
rules.

Table 5 shows the previous and newly implemented rules.

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
<th>Source</th>
<th>Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unnamed entities/relationships</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>2</td>
<td>Identically-named entities/relationships</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>3</td>
<td>Dangling edges</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>4</td>
<td>Type violations</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>5</td>
<td>Multiple inheritance</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>6</td>
<td>Parallel edges (strict)</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>7</td>
<td>Parallel edges (loose)</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>8</td>
<td>Cyclic inheritance</td>
<td>Current</td>
<td>Previously Implemented</td>
</tr>
<tr>
<td>9</td>
<td>Inherited cyclic composition inconsistency</td>
<td>[6,13]</td>
<td>Newly Implemented</td>
</tr>
<tr>
<td>10</td>
<td>* Disconnected element</td>
<td>[6,13]</td>
<td>Newly Implemented</td>
</tr>
<tr>
<td>11</td>
<td>* Unreachable element</td>
<td>[6,13]</td>
<td>Newly Implemented</td>
</tr>
<tr>
<td>12</td>
<td>Type violation (Edge links two attributes)</td>
<td>[26]</td>
<td>Newly Implemented</td>
</tr>
<tr>
<td>13</td>
<td>Attribute name checking</td>
<td>[14]</td>
<td>Newly Implemented</td>
</tr>
<tr>
<td>14</td>
<td>Attribute name in a class</td>
<td>[26]</td>
<td>Newly Implemented</td>
</tr>
</tbody>
</table>

Table 5. Rule Implementation

The rule description is as follows:

1. **Unnamed entities/relationships**
   This rule returns entities and relationships without names. There can be some entities and
   relationships without names in the models and it can cause inconsistency for the whole model.
   When an entity does not have the name then it cannot be compared to other entities with same
   roles, so that reduces the consistency of the whole model.

2. **Identically-named entities/relationships**
   The rule returns distinct entities and relationships that have identical names. If any entities/
   relationships have same names then they cannot be correctly mapped and unified in the whole
   model.

3. **Dangling edges**
   “Occurs when a parameter’s or attribute’s type refers to a class that does not exist in the model”
   [11]. Within this rule both endpoints of an edge must be connected to nodes. In case of having
disconnected edges, the whole model will be inconsistent.
4. **Type violations**  
The rule checks for the compatibility of the type of an edge with the types of its endpoints. In case of having violation the whole model will be inconsistent as well.

5. **Multiple inheritance**  
The rule will check the objects not to have more than one parent. In case of multiple parents the result will be the inconsistency in the whole model.

6. **Parallel edges (strict)**  
In case of having parallel edges of the same type and directionality, it indicates a problem and inconsistency.

7. **Parallel edges (loose)**  
In case of having parallel edges of arbitrary types and directionality, there may be indicative of a problem and inconsistency.

8. **Cyclic inheritance**  
The rule checks for the inconsistencies related to the cycles in the generalization hierarchy.

9. **Inherited cyclic composition inconsistency**  
The rule checks for the composition relationships. The inconsistency, arises when a composition relationship (the multiplicity constraints of this composition relationship are important) and inheritance relationships between classes of a model form a cycle that produces an infinite containment of the instances of the affected classes [6 and 13].

10. **Disconnected element***  
The rule checks for the classes which are disconnected, i.e., no incoming or outgoing edge. This can be stated as a multiplicity violation rule [6 and 13].

11. **Unreachable element***  
The rule checks for the classes that are not reachable via navigable edges. For example, if we have 3 classes A, B, C and only have A->B<-C, then A and C are inconsistent unless they are declared as "root" [6, 13].

12. **Type violation (Edge links two attributes)**  
The rule checks for the edges which are between two attributes. All attributes should be connected to the entities and there should be no edges between two attributes.

13. **Attribute name checking**  
The rule checks for the classes which use the same attribute names as outgoing association end names [14].

14. **Attribute name in a class**  
For the consistency in the whole model no two attributes may have the same name within a class and the rule checks for such inconsistencies [14].

15. **Parent class attribute**  
In a consistent model, parent class attribute should not refer to child class, so the rule check for such problems.

* We check for all classes X, either X or one of X's parents should be reachable via association edges.
4. Case Study

In this chapter, we discuss two case studies conducted to validate our methodology for GCC. There are two different problem domains in the study; a Warehouse system and a Hospital system. Models, which were used in the case studies collected from Simula Research Laboratory (Norway). These models include conceptual models indicated by UML class diagram.

These two problem domains involve human subjects and natural circumstances because of having different stakeholders with different requirements and concerns over the system development. Therefore, these problem domains motivate us to choose them in order to explore and understand multiple viewpoints of different stakeholders for a whole system, which is distributed between development teams in an MDE approach, i.e. having multiple models for a whole system in order to model these problem domains as a fragment of OOA.

These two problem domains have different stakeholders, which result in having different models, e.g., several wards, display units, medical teams as physicians, technicians (technical staff), nurses, administrative assistants and patients in Hospital and marshalling area, shipping, ordering, picking, etc. in Warehouse.

Having numbers of models for a whole system, including multiple stakeholders, developers and viewpoints cause to rise conflicting and even contradicting models’ elements, concepts, etc., all of which are considered as inconsistencies between these models. To handle consistency problems and ensure a consistent system model, consistency checking must be done as a basic activity in consistency management.

It should be mentioned, both selected cases have large number of overlapping-elements, which resulted in having more complications and challenges for resolving the diagnosed inconsistencies. It helps us to evaluate GCC methodology in more details. Whereas, we might have a probability of less overlapping-elements, and consequently, less complexity in other cases.

The inconsistencies are expressible by the semantic of UML class diagram. It means that, the inconsistencies were detected against consistency checking rules and the rules are defined apart from the specific problem domain e.g., hospital, warehouse, etc.

As it is mentioned in Section 3.1, two systems of models are defined and studied in this thesis. The purpose of using two strategies (systems) was to build more confidence in that GCC methodology remains feasible, irrespective of the interconnection strategy chosen.

According to Section 3.1, we implemented ‘One model against (n-1) models’ strategy on case study 1 (Warehouse) and ‘Star models’ strategy on case study 2 (Hospital). Then, all connected models were mapped together. For both case studies, we provided all the mappings automatically by name matching [49]. As the next phase, all models were merged together and made one whole model. The models were checked against consistency checking rules. According to the detected inconsistencies, we tried to resolve them and improve the relationships.
We describe these two case studies as follow;

4.1. Case Study1: Warehouse

4.1.1. Data Collection

A warehousing company has a system for handling orders. The system tracks the status of the orders, provides computer support for warehousing workers, and keeps track of inventory levels in the warehouse. The orders are for minivan bumper fascia (covers for bumpers) in various colors. The warehouse has different zones and floors. Warehouse pickers work in different floors and load the fascias into pallets and consequently into trucks. The handling of selection and loading of fascia for bumpers will be done by the warehouse. They sort and deliver pallets of fascia to the automotive factory. So that, the factory can use them for direct loading to assembly line. The most important activity is to provide these sorting and delivering in a complete correct order. In case of having a truck in the automotive factory, with an even one fascia in an incorrect order in the truckload, the error holds up to assembly line and huge amount of financial loss will be resulted for the warehousing company. More details are available in Appendix A.

For modeling this problem domain, sixteen models (UML class diagram) have been provided based on stakeholders’ needs and concerns, which are used in our study. The way that the models have been gathered is defined in Section 4.1.2.

4.1.2. Model Designs

The stakeholders were not skilled enough to provide a complete, correct and consistent model for the related system. Source of information for the system included different viewpoints elicited from different stakeholders, which were in separated groups of models (raw models) for each part of the system (See Appendix B). For instance, Figure 7 to Figure 12 show different parts of Warehouse system defined by one of the stakeholders. These parts are Order, Picking, Sequencing, Loading and Replenishing. Figure 7 shows the class view and Figure 8 shows the entity-relation view of Order. These two diagrams have been shown for better understanding of all models which will be defined, discussed and shown in other parts of the thesis. Figure 9 to Figure 12 indicate the entity-relation views of the sample for Warehouse.

![Figure 7. Part1-Order (Class view)](image-url)
Figure 8. Part1-Order (Entity relation view)

Figure 9. Part2- Picking

Figure 10. Part3- Sequencing
As illustrated in Figure 8 to Figure 12, the models have some common entities and attributes, for instance *Fascia* with its attribute *SKU*. Figure 13 shows the whole model (of those different parts), which we have made based on the commonalities and differences.
According to Figure 8 to Figure 12, this stakeholder has defined five different parts of Warehouse system that had many common UML entities and relationships incorporated in the model in Figure 13. Table 6 shows the number of elements within different parts of this viewpoint of the stakeholder. The last row of Table 6 shows the number of entities of the final model which is a whole view for this stakeholder.

Table 6. UML Element count of Different Parts of a Model

<table>
<thead>
<tr>
<th>Part #</th>
<th>Element count</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node</td>
<td>Edge</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>9</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>18</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>13</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>49</td>
<td>50</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>Final</td>
<td>32</td>
<td>36</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. UML Element count of Different Parts of a Model

For Warehouse system, all sixteen models have different parts which have been treated the same as Figure 8 to Figure 13 in order to provide the whole models. The same set of activities has been done for all eight models in Hospital (case study 2).

Therefore, the mentioned set of activities is considered as a ‘pre-requisite step’ for consistency checking by GCC. This step makes all models in UML Class-diagram notation, including attributes (if they have) and without methods. Each model should represent each stakeholder’s needs and concerns for the whole system (making all different parts unified in one model for each stakeholder). These models, due to lack of an automatic transformer for TreMer\textsuperscript{+}, should be manually transferred to TreMer\textsuperscript{+}, and checked for syntactic and semantic correctness.

It should be mentioned that limitations which are described by the pre-requisite step come from lack of strong tools support for providing consistency checking by GCC. The reason for the limitations is that GCC is a recent idea and it needs more evaluation from different points of view. Lack of tool support is one of the aspects should be worked on to generalize GCC in e.g., an industrial scale. Therefore, the
limitations for having simple UML class-diagrams (including attributes, excluding methods) belong to TreMer+ (as a limited tool support), and not for the idea of GCC.

4.1.3. Case Study Conduct

In this problem domain, sixteen different models were developed based on different stakeholders’ needs and concerns. In order to make a whole model that would be complete, correct and consistent, we check the consistency between them by our methodology that is described, as follows;

1. All models are provided in class diagrams.
2. All individual models are checked for their correctness, according to the requirements. In this step, all models are checked in order to compare between what the stakeholder has defined and what a class diagram has been designed.
3. A strategy has to be selected for defining the initial system of models in order to connect our different models. In Warehouse system, the strategy is “One against (n-1) models” (See Figure 14).

![Figure 14. One Model against (n-1) Models- (16 models)](image)

These 16 models are as V1 to V16. For implementing the connection between them and defining the mappings, model V1 is selected. Since the models overlapped a lot, using a "base model" for establishing the relationships is advantageous in the sense that a large fraction of the element correspondence could be readily identified without the need for building a connector between the models. Instead, the connector, we build later, focuses on the relationships that cannot be captured via the "base model". For the choice of the base model, best results (in terms of identifying the maximum number of element correspondences) will be achieved, if we choose the most complete model as a basis model.

It should be mentioned that, we did not check the models for completeness beforehand and simply chose V1. In fact, any of the other 15 models could be used as the basis one. Figure 15 shows V1 in the first step.
Figure 15. Model V1 in First Steps

According to the types of relation such as, simple Directed link, Composition, Generalization, Association, etc., the entities are connected with the specific edges in each model. Table 7 defines different relations between different entities and attributes. It should be mentioned that, some entities have attributes and some others have not.

<table>
<thead>
<tr>
<th>Relationship Symbol</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>Directed link(Association)</td>
<td>Show the relation between instances of classes.</td>
</tr>
<tr>
<td>—</td>
<td>Generalization</td>
<td>A relationship in which one model element (the child) is based on another model element (the parent).</td>
</tr>
<tr>
<td>▼</td>
<td>Aggregation(Composition)</td>
<td>Shows a classifier as a part of or subordinate to another classifier.</td>
</tr>
<tr>
<td>—</td>
<td>Link</td>
<td>A relationship between objects or instances of the classifiers or nodes.</td>
</tr>
</tbody>
</table>

Table 7. Relation Description

Figure 16 shows UML class diagram for the class Picking Request with two attributes Pallet and Picking route. The class diagram has three parts as class name, attributes and methods, but the methods were not included in this study. As Figure 17 indicates that, in our models all entities, which stand for the classes, have been shown with rectangles and all the attributes have been shown with ovals.
4. In the next step, according to Figure 14, the mapping of V1 and other 15 models is provided by the automated mapping based on name matching and with the usage of N-gram algorithm with N=2 [49]. Through the changing threshold value, the matcher can handle variations in spelling and words, for instance, ‘PickingRequest’ (without space between Picking and Request) and ‘Picking Request’ (with a space between Picking and Request) will not be matched with the exact name matching, but by using N-gram and a threshold value of 0.9, these two words will be matched. After this matching, all 15 models will be mapped with V1. The matching includes entities, attributes and relations as well.

5. The next step is providing a merged model of the whole sixteen models which have been mapped together. Here is the element count of the resulted merged model;

**Element counts:**
- Node count: 518
- Edge count: 749
- Total: 1267

6. After merging the models, the consistency will be checked against the consistency rules. It means that, all relations and entities will be checked.

7. Inconsistencies can arise from more than two models. It should be mentioned that, such an ability of detecting such inconsistencies provides the advantage of GCC (e.g., in comparison with PCC). Figure 18 shows a sample of the rules named ‘Multiple parents’ and the diagnosed report, which stands for the inconsistency between more than two models, for instance, in Merge A, the object with id number 34891 has multiple parents as 35069, 35098, 35150, 3525 and 35258.
In the first round of consistency checking of the merged model, detected inconsistencies are as follows:

a. Identically-named entities and relationships
b. Identically-named attributes within individual entities and relationships
c. Objects with multiple parents
d. Parallel edges (typed)
   i. Parallel Aggregation edges
   ii. Parallel DirectedLink edges
   iii. Parallel Generalization edges
   iv. Parallel Link edges
e. Parallel edges (un-typed)

4.1.4 Resolution Approach

Six different iterations have been done during resolution approach, each of which followed by different systematic procedures for resolving inconsistencies and ended with merging models. Each of iteration and the related systematic procedure are a contribution for the thesis which is defined and provided for resolving the inconsistencies. Although, it has basically used in the previous methodology in [1], different resolution’s procedures have been defined and used here.

A final consistent model is provided in final step. The answer of RQ3 and RQ4 is also shown in these steps and also detailed discussed in Section 5. Here all related iterations and their effects on the models are defined;

1. Merge1- Connector (identically named entities)

The first group of inconsistencies, Identically-named entities and relationships, concerns those elements visible in different models which have the same names, however, not mapped in the overall model. When we merge the models, inconsistency problems occur between elements. Also, it should be mentioned that, this inconsistency is regarded to the entities that are not as common entities between V1 and other 15 models. In other words, these entities are common between other models rather than V1. So, although we have matched model V1 with other models, these entities need to be mapped between their related models.

As a result, the procedure for resolving such an inconsistency is to map those entities and make them as the same entities between the models that have initially used such entities. For this reason, a connector is defined in order to put these entities in, and map the connector with other 16 models. Figure 19 shows the connector between the models (V1 to V16).
Figure 19. Connector and Matching

Figure 20 is a close snapshot of Figure 19 for better understanding of the mapping between the connector and the models.

Figure 20. Connector

Figure 21 indicates $MapC2$ between the connector and V2, as a sample.
All the entities, which are inconsistent against the rule *identically named entities*, are putted in the connector in order to be mapped between the related models. As a sample of related inconsistency, an entity named *Order* is common between V2, V3, V4, V5, V6, V7, V10, V11 and V12. So, we made an entity with the same name *Order* in the connector and mapped the connector with the mentioned models. As illustrated in Figure 21, the entities as *Office, System, PickingRequest* and *TranslationTable* have the same commonality and they are putted in the connector. The inconsistent entities are made in the connector and the connector is mapped to the models.

**2. Merge2- Connector (Identically-named attributes within individual entities and relationships)**

After finishing the mapping between the connector and the models, a merged model is made named *Merge 2*. The whole model is checked again, for detecting inconsistencies. The next group of inconsistencies diagnosed is *Identically-named attributes within individual entities and relationships*. These inconsistencies are detected due to the same name attributes of the entities which are common between different models. It means that, although the entities are mapped, still their attributes with the same names cause the merged model to be inconsistent. So, in this step, the attributes, which had such a problem, are added to the related entity in the connector. Then, they are mapped to the related models. Figure 22 shows two attributes of *order* entity named *model* and *color* which are multiple in the merged model (Merge A2) and come from different un-mapped models (e.g., V12). Figure 23 shows the related attributes made in the connector and mapped with V12, as illustrated, *model* and *color* attributes are added to *order* entity in the connector.
3. **Merge3- (Objects with multiple parents)**

In the next step of consistency checking, for the new merged model (Merge3), resolution’s procedure is focused on the inconsistency objects with multiple parents. Figure 24 shows Back Fascia entity with multiple parents as Fascia and AbstractFascia.
The procedure, for resolving objects with multiple parents inconsistency, is to map the multiple parents entities between the models (that have those entities) and the connector. Figure 25 shows the mappings for resolving this inconsistency. In this figure, Fascia in the connector is mapped between Fascia in V16 and AbstractFascia in V14.

![Figure 25. Multiple parents, Inconsistency resolving](image)

This mapping is also done for other models that have the same inconsistency problem. The important point is that, we map not only the source and target entities of multiple parent relations, but also the edges of them as well. So, by the next round of merging, multiple parent inconsistency will be resolved.

4. Merge4- (Parallel edges (typed))

After resolving the previous group of inconsistencies, all models are merged and the resulted model, i.e., Merge4, is checked. The next group of inconsistencies is Parallel edges (typed), which stands for the case of having parallel edges of the same type. There are four types of parallel edges in the model as, Parallel aggregation edges, Parallel directed link edges, Parallel generalization edges and Parallel link edges. Figure 26 shows the parallel aggregation edges between Reserve Room and Warehouse which are caused by V6, V7, V9 and V15.
For resolving this type of inconsistency, the edges are defined in the connector between the related entities and mapped with the related models. Figure 27 indicates the mapping for this aggregation edge between the connector and the related models.

5. Merge5- (Parallel edges (un-typed))

The models, resulted from the previous resolving procedures, are merged and the new model (Merge5) checked against the consistency rules. The last group of inconsistencies is the parallel edges (un-typed), which further implies to the case of having parallel edges of arbitrary types and directionalities. Figure 28 indicates such an inconsistency between the entities Picking Request and Order, as shown, there are four parallel edges between them caused by V2, V6, V11, V16.
The procedure is to make expert judgment to choose the best alternative amongst the alternatives coming from different models. The selection is in accordance with the best definition and relationship made in the model that has the most compatibility with the information elicited from the stakeholder. In better words, the selection of the best relation is logically regarded to requirements and correctness of this relationship. As illustrated in Figure 28, the aggregation relationship is selected for this edge as the best choice between order and picking request. This relationship (aggregation) is defined in the connector and mapped with the models V2 and V11 which have the same type of relationships. All the other relationships between order and picking request which came from other models are treated as refuted (See pages 8 to 15 in [5]). Note that, the refuted elements are ignored during consistency checking as if they do not exist. Figure 29 shows the refuted edges in related models, V6 and V16, and the connector’s entity relationship for these edges (The refuted elements are colored, in magenta, in Figure 29).
It should be mentioned that, in case of refuting all edges of an entity in a model, the entity is refuted as well. This procedure is due to not having any disconnected element, which is checked by the rule *disconnected element*.

6. **Merge6- (Refuted elements)**

The next merged model, named Merge6, is checked and all parallel edges are completely resolved. But due to the refuted elements, the whole model (merge6) is not a clear and concise model. It means that, according to not existence (repudiated) of some entities, we have some dangling edges. For such entities, we have to remove those edges form the related models.

As the final step, all the refuted entities and relationships are eliminated from the related models, therefore, the final merged model (Merge7) is clear and consistent. Although, the
refuted elements are deleted from the models, we achieve a consistent merged model because of all the previous resolution procedures. In better words, the elimination of the entities does not affect the consistency of the merged model. The elements counts of the final consistent merged model are as follows;

**Element counts:**
- Node count: 287
- Edge count: 399
- Total: 686

4.1.5. Analysis and Results

As we aimed to make GCC operational and applicable according to the RQs, we have undertaken two case studies in this thesis in order to provide more definitive answers to whether or not GCC is feasible and useful. So, in the first case study, the required data for answering RQ1, RQ2, and RQ3 are provided as follows;

As discussed in Section 4.1.4, Table 8 represents different merge iterations of Warehouse models, with the number of inconsistencies in different groups. The previous group of inconsistencies is resolved and the new merged model is defined in each iteration, as follows;

- ‘Merge1’ is the first merge of the models,
- ‘Merge2’ is resolved for *Identically-named entities* inconsistency,
- ‘Merge3’ is resolved for *Identically-named attributes* inconsistency,
- ‘Merge4’ is resolved for *Multiple parents* inconsistency,
- ‘Merge5’ is resolved for *Parallel edges resolved (typed)* inconsistency,
- ‘Merge6’ is resolved for *Parallel edges resolved (Un-typed)* inconsistency,
- ‘Merge7’ is the final merged model.
In Table 8 shows all number of inconsistencies and UML elements. So, by each iteration, the previous number of inconsistencies has changed to 0.

<table>
<thead>
<tr>
<th>Inconsistency Type</th>
<th>Merge1</th>
<th>Merge2</th>
<th>Merge3</th>
<th>Merge4</th>
<th>Merge5</th>
<th>Merge6</th>
<th>Merge7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unnamed nodes</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Identically-named entities and relationships</td>
<td>107</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Disconnected nodes</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Identically-named attributes within individual entities</td>
<td>6</td>
<td>50</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>and relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attributes named identically to links within individual</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>entities and relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attributes whose type is inherited from their owning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>entity / relationship</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dangling edges</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Type violations</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Objects with multiple parents</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallel edges (typed)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Aggregation edges</td>
<td>1</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parallel DirectedLink edges</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parallel Generalization edges</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parallel Link edges</td>
<td>12</td>
<td>32</td>
<td>32</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parallel Similarity edges</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parallel edges (un-typed)</td>
<td>27</td>
<td>66</td>
<td>66</td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Objects on cyclic inheritance paths</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Objects on cyclic composition paths</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| Total number of inconsistencies in each iteration       | 167    | 179    | 129    | 103    | 87     | 0      | 0      |

<table>
<thead>
<tr>
<th>Number of UML elements</th>
<th>Node</th>
<th>Edge</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>518</td>
<td>749</td>
<td>1267</td>
</tr>
<tr>
<td></td>
<td>396</td>
<td>745</td>
<td>1141</td>
</tr>
<tr>
<td></td>
<td>354</td>
<td>702</td>
<td>1056</td>
</tr>
<tr>
<td></td>
<td>320</td>
<td>515</td>
<td>835</td>
</tr>
<tr>
<td></td>
<td>318</td>
<td>504</td>
<td>822</td>
</tr>
<tr>
<td></td>
<td>272</td>
<td>374</td>
<td>646</td>
</tr>
<tr>
<td></td>
<td>287</td>
<td>399</td>
<td>686</td>
</tr>
</tbody>
</table>

Table 8. Inconsistencies in Each Stage

As illustrated in the last two rows of Table 8, Total number of inconsistencies and Number of UML elements are decreased in each iteration. It means that, GCC enables us to incrementally resolve inconsistencies and achieve the consistent merged model.

According to four different types of resulted inconsistencies, Table 9 shows both global and pair-wise inconsistencies. Therefore, GCC enables to find inconsistencies, which are hard to detect or not detectable at all by using PCC because of rooting from more than two models.
<table>
<thead>
<tr>
<th>Type of inconsistency</th>
<th>Number of inconsistencies</th>
<th>Rate %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global</td>
<td>Pairwise</td>
</tr>
<tr>
<td>Identically-named entities and relationships</td>
<td>66</td>
<td>41</td>
</tr>
<tr>
<td>Identically-named attributes within individual entities and relationships</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Objects with multiple parents</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Parallel edges</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>Parallel edges (un-typed)</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>94</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 9. Warehouse Inconsistencies

Additionally, GCC scales different models with e.g., different number of elements (Model Size). Table 10 shows element counts of sixteen models of Warehouse during merging with each other.

<table>
<thead>
<tr>
<th>Node count</th>
<th>Edge count</th>
<th>Total Element count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Model</td>
<td>39</td>
<td>54</td>
</tr>
<tr>
<td>2 Models</td>
<td>79</td>
<td>118</td>
</tr>
<tr>
<td>3 Models</td>
<td>116</td>
<td>167</td>
</tr>
<tr>
<td>4 Models</td>
<td>140</td>
<td>216</td>
</tr>
<tr>
<td>5 Models</td>
<td>168</td>
<td>266</td>
</tr>
<tr>
<td>6 Models</td>
<td>196</td>
<td>299</td>
</tr>
<tr>
<td>7 Models</td>
<td>256</td>
<td>371</td>
</tr>
<tr>
<td>8 Models</td>
<td>274</td>
<td>391</td>
</tr>
<tr>
<td>9 Models</td>
<td>306</td>
<td>430</td>
</tr>
<tr>
<td>10 Models</td>
<td>324</td>
<td>468</td>
</tr>
<tr>
<td>11 Models</td>
<td>339</td>
<td>495</td>
</tr>
<tr>
<td>12 Models</td>
<td>368</td>
<td>531</td>
</tr>
<tr>
<td>13 Models</td>
<td>393</td>
<td>566</td>
</tr>
<tr>
<td>14 Models</td>
<td>430</td>
<td>605</td>
</tr>
<tr>
<td>15 Models</td>
<td>475</td>
<td>676</td>
</tr>
<tr>
<td>16 Models</td>
<td>518</td>
<td>749</td>
</tr>
</tbody>
</table>

Table 10. Element Counts of 16 Models

Table 11 shows the estimated effort required for resolving each group of inconsistencies against each consistency rule according to Resolution Approach (See Section 4.1.4). Based on RQ3, we have shown the feasibility of incrementally resolving the inconsistencies.

<table>
<thead>
<tr>
<th>Inconsistency resolving stages</th>
<th>Effort (person-hours)</th>
<th>Number of inconsistencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identically-named entities and relationships</td>
<td>10</td>
<td>107</td>
</tr>
<tr>
<td>Identically-named attributes within individual entities and relationships</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>Objects with multiple parents</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Parallel edges (typed)</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Parallel edges (un-typed)</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Refuted elements</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>19</td>
<td>176</td>
</tr>
</tbody>
</table>

Table 11. Effort level for Each Consistency Resolving Stage
Figure 30 to Figure 43 show different elapsed time (second) of checking each consistency rule for the 16 models. The time of consistency checking has a linear boost against each rule.

Figure 30. Unnamed Nodes

Figure 31. Identically-named Entities and Relationships

Figure 32. Disconnected Nodes
Figure 33. Distinct Attribute Names

Figure 34. Disjoint Name sets for Attributes and Links

Figure 35. Navigable Nodes
Figure 36. Distinct Attribute types and Inherited types

Figure 37. Dangling Edges

Figure 38. Type Violation
Figure 41. Parallel Edges (loose)

Figure 42. Cyclic Inheritance

Figure 43. Cyclic Composition
According to the illustrated figures, all time durations for consistency checking have the same linear boost approximately. It means that, in GCC, with increasing number of models, the required time for consistency checking increased as well. Figure 44 indicates all diagrams of the rules together for better illustration of the result.

Figure 44. All Consistency checking Rules Diagrams
4.1.5.1. Value Categorization of Inconsistencies

The inconsistencies could be categorized from different points of view, one of the important is ‘value categorization’ that focuses on the risk of the inconsistencies. We aimed to study if an inconsistency was not detected, to what extent it would result in problem(s) and vise versa. We have defined three groups, respectively named Trivial, Moderate and Important. Table 12 indicates the categories and time duration for each type of inconsistencies in our study.

<table>
<thead>
<tr>
<th>Type</th>
<th>Trivial</th>
<th>Time (Sec.)</th>
<th>Type</th>
<th>Moderate</th>
<th>Time (Sec.)</th>
<th>Type</th>
<th>Important</th>
<th>Time (Sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null rule</td>
<td></td>
<td>1,907</td>
<td>Unnamed entities/relationships (Unnamed Nodes)</td>
<td>2,079</td>
<td></td>
<td>Dangling edges</td>
<td></td>
<td>2,334</td>
</tr>
<tr>
<td>All nodes</td>
<td></td>
<td>2,044</td>
<td>Identically-named entities/relationships</td>
<td>2,281</td>
<td></td>
<td>Multiple inheritance</td>
<td></td>
<td>2,365</td>
</tr>
<tr>
<td>All edges</td>
<td></td>
<td>2,104</td>
<td>Type violations</td>
<td>3,020</td>
<td></td>
<td>Parallel edges (strict)</td>
<td></td>
<td>2,897</td>
</tr>
<tr>
<td>Un-Navigable nodes</td>
<td></td>
<td>1,882</td>
<td>Disconnected elements</td>
<td>2,390</td>
<td></td>
<td>Parallel edges (loose)</td>
<td></td>
<td>2,773</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Attribute name checking (Distinct attribute names)</td>
<td>2,212</td>
<td></td>
<td>Cyclic inheritance</td>
<td></td>
<td>1,972</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Inherited cyclic composition inconsistency (Cyclic composition)</td>
<td>1,961</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Parent class attribute(Distinct attribute types and inherited types)</td>
<td>2,032</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>7,937</td>
<td></td>
<td>14,195</td>
<td></td>
<td></td>
<td></td>
<td>16,334</td>
</tr>
</tbody>
</table>

Table 12. Value Categorization of Inconsistencies for Warehouse

Table 12 refers to the extent of problems could be occurred if the inconsistencies are not detected. For instance, if GCC does not diagnose one of the inconsistencies of ‘Important’ group, detecting and resolving unexpected effects would be complicated in terms of time duration, human source, effort-level, cost, etc. Whereas, ‘Moderate’ ones could result in less complexity in case of not detecting and finally, ‘Trivial’ group would not cause a major problem for the following processes. Another category is regarded to ‘False positive’ and ‘False negative’ errors (inconsistencies).

4.1.5.2. False Positive

There are some types of inconsistency which are mainly related to the defined consistency rules in GCC methodology and TreMer+. This group of inconsistencies is about statistical data, tool’s configuration (debugging) and the structure of model designs. In better words, this group of inconsistencies is not really as inconsistencies.

It includes these types of inconsistency, as follows;

- **Null Rule** (debug): To obtain an RML translation of the view being checked.
- **All nodes** (debug): It results in counting the number of nodes.
- **All edge** (debug): It results in counting the number of edges.
- **Un-navigable nodes** (model design): Each entity and relationship should have at least one incoming edge.
For this group of inconsistencies, they all prolonged the time duration of consistency checking, whereas in case of not selecting the rules to be checked, time could decreased apparently. Table 13 indicates the required time for checking related rules for 16 models in Warehouse.

<table>
<thead>
<tr>
<th>Inconsistency Type</th>
<th>Time duration (Sec)</th>
<th># of Inconsistencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null rule</td>
<td>1.907</td>
<td>0</td>
</tr>
<tr>
<td>All nodes</td>
<td>2.044</td>
<td>518</td>
</tr>
<tr>
<td>All edges</td>
<td>2.104</td>
<td>749</td>
</tr>
<tr>
<td>Un-Navigable nodes</td>
<td>1.882</td>
<td>29</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7.937</strong></td>
<td><strong>1296</strong></td>
</tr>
</tbody>
</table>

Table 13. False Positive Inconsistencies for Warehouse

### 4.1.5.3. Correct Inconsistencies

This group of inconsistencies stands for the inconsistencies which are correct and detected by the tool. Table 14 shows the number of the inconsistencies (in different types) of ‘Correct group’ in accordance to the results.

<table>
<thead>
<tr>
<th>Inconsistency Type</th>
<th># of Inconsistencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identically-named entities and relationships</td>
<td>107</td>
</tr>
<tr>
<td>Identically-named attributes within individual entities and relationships</td>
<td>6</td>
</tr>
<tr>
<td>Objects with multiple parents</td>
<td>4</td>
</tr>
<tr>
<td>Parallel edges (typed)</td>
<td>23</td>
</tr>
<tr>
<td>Parallel edges (un-typed)</td>
<td>27</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>167</strong></td>
</tr>
</tbody>
</table>

Table 14. Correct groups of Inconsistency in Warehouse

Total number of the inconsistencies in the first step of GCC is 1296+167=1463. Additionally, the rules ‘Null rule’, ‘All nodes’ and ‘All edges’ are related to tool debugging rather than to consistency checking. But the rule ‘Un-Navigable’ is related to the structure of model designs. As a result, the rate of ‘False positive’ inconsistencies is 1.98%, as follows;

\[
\text{Un-Navigable nodes} \div \text{Total} = \text{rate of False positive \%} \\
29 \div 1463 = 0.019 \times 100 = 1.98\%
\]

### 4.1.5.4. False Negative

In this thesis, there are some types of inconsistency, which are occurred due to naming errors of entities or relations. The only group of such an inconsistency is, as follows;

**Identically-named entities and relationships**: Though, all entities and relationships are checked by the related rule, still there are some **Identically-named entities and relationships** inconsistencies, which are not detected by the tool due to different naming types by stakeholders. For instance, one has named an entity as ‘A’, whereas, another has named ‘B’, although they both imply the same entity. For such inconsistencies, we have defined ‘Manual inspection’ phase in GCC. In this group of inconsistencies, it is needed to manually detect such types of error. Table 15 shows required activities for detecting 5 numbers of ‘False negative’ inconsistencies with their time duration. They are related to the entities which are the same in the models but they are defined differently by the stakeholders. ‘False negative’ inconsistencies in the study are as follows;

1. Rear Fascia, Back Fascia
2. Team, Employee, Employee member, Warehouse Workers
3. Control System and Administrative System
4. Touch Screen, Interactive display, Screen
5. List of Picking Request, Pick-Up Queue

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time duration (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>1800</td>
</tr>
<tr>
<td>Merge Computation</td>
<td>0.038</td>
</tr>
<tr>
<td>Merge visualization</td>
<td>0.481</td>
</tr>
<tr>
<td>Consistency Checking</td>
<td>30.748</td>
</tr>
<tr>
<td>New Detection</td>
<td>300</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2131.267</strong></td>
</tr>
</tbody>
</table>

Table 15. False Negative

The main problem for detecting such an inconsistency is the tool inability to provide a proper visualization for final inspection. TreMer+ is incapable of providing the final tangible view for the detection. This will impose more effort for final inspection step of GCC methodology.

This type of analysis can lighten the way for coming studies in future on GCC, in particular, in more human centered and controlled experiments. So, it would provide the ability to study GCC in terms of scalability, performance, cost, resource demands, etc.

4.2. Case Study2: Hospital

4.2.1. Data Collection

A hospital requires a system for providing automated services. The system handles the services in the hospital with different wards and staffs for inpatients and outpatients. The hospital has wards and rooms with different beds. The identification of a bed is regarded to ward’s, room’s and bed’s number. All equipments in a ward can be mobile or stationary. For mobile equipment, a booking calendar keeps track of when and where the piece is needed. The set of units in different wards varies for each ward. The display and scanner of wards and rooms show everything from a medical team member’s schedule to a patient's history. There is a unique id for each medical team members. Also, all patients are grouped as Inpatient and/or Outpatient (More details are available in Appendix A).

For this problem domain, eight UML class diagram models are provided based on stakeholders’ needs and concerns, which are used in the study.

4.2.2. Model Designs

The description of model design is the same as the first case study (See Section 4.1.2).

4.2.3. Case Study Conduct

For this problem domain, the consistency checking is done as follows;

1. All models are provided in class diagrams.
2. All individual models are checked for their correctness, according to the requirements. In this step, all models are checked in order to compare between what the stakeholder has defined and what a class diagram has been designed.
3. A strategy has to be selected for defining the initial system of models in order to connect our different models. In Hospital system, the strategy is “Star” (See Figure 45).
4. In the next step, according to Figure 45, the mapping of all eight models is provided by the automated mapping based on name matching and with the usage of N-gram algorithm with $N=2$ [49]. In this step, the threshold for the mapping is defined as 0.9. After this matching, all eight models will be mapped. The matching includes entities, attributes and relations as well.

5. The next step is providing a merged model of the whole eight models which have been mapped together. Here is the element count of the resulted merged model;

   **Element counts:**
   - Node count: 120
   - Edge count: 268
   - Total: 388

6. After merging the models, the consistency will be checked against the consistency rules. It means that, all relations and entities will be checked.

7. As mentioned in the first case study, inconsistencies can arise from more than two models. In the first round, detected inconsistencies are as follows;
   a. Objects with multiple parents
   b. Parallel edges (typed)
      i. Parallel DirectedLink edges
      ii. Parallel Link edges
   c. Parallel edges (un-typed)

### 4.2.4. Analysis and Results

According to the four different resulted inconsistencies, GCC enables us to detect both global and pair-wise inconsistencies, as it is shown in Table 16.

In fact, the aim for this case study is to find the results of having GCC with another type of ‘Initial system of models’ (i.e., “Star” strategy) with using a new problem domain. It means that, we preferred not to be affected by the learn-ability of the problem domain for better studying on GCC.
4.2.4.1. Value Categorization of Inconsistencies

Table 17 refers to the extent of problems could be occurred if the inconsistencies are not detected.

<table>
<thead>
<tr>
<th>Type of Inconsistency</th>
<th>Number of Inconsistencies</th>
<th>Rate %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global</td>
<td>Pairwise</td>
</tr>
<tr>
<td>Objects with multiple parents</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Parallel edges (Parallel Directed Link edges)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Parallel edges (Parallel Link edges)</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Parallel edges (un-typed)</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>14</strong></td>
<td><strong>25</strong></td>
</tr>
</tbody>
</table>

Table 16. Hospital Inconsistencies

<table>
<thead>
<tr>
<th>Type of Inconsistency</th>
<th>Time duration (Sec)</th>
<th># of Inconsistencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null rule</td>
<td>11,542</td>
<td>0</td>
</tr>
<tr>
<td>All nodes</td>
<td>0,721</td>
<td>120</td>
</tr>
<tr>
<td>All edges</td>
<td>0,797</td>
<td>268</td>
</tr>
<tr>
<td>Un-Navigable nodes</td>
<td>0,852</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>13,912</strong></td>
<td><strong>406</strong></td>
</tr>
</tbody>
</table>

Table 17. Value Categorization of Inconsistencies for Hospital

4.2.4.2. False Positive

Table 18 shows ‘False positive’ errors (inconsistencies) detected in this case study. The type of ‘False positive’ errors in this study are the same as Warehouse study. Table 18 indicates the required time for checking related rules for 8 models in Hospital study.

<table>
<thead>
<tr>
<th>Inconsistency Type</th>
<th>Time duration (Sec)</th>
<th># of Inconsistencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null rule</td>
<td>11,542</td>
<td>0</td>
</tr>
<tr>
<td>All nodes</td>
<td>0,721</td>
<td>120</td>
</tr>
<tr>
<td>All edges</td>
<td>0,797</td>
<td>268</td>
</tr>
<tr>
<td>Un-Navigable nodes</td>
<td>0,852</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>13,912</strong></td>
<td><strong>406</strong></td>
</tr>
</tbody>
</table>

Table 18. False Positive Inconsistencies for Hospital
4.2.4.3. Correct Inconsistencies

Table 19 shows the number of the inconsistencies (in different types) of ‘Correct group’ in accordance to the results.

<table>
<thead>
<tr>
<th>Inconsistency Type</th>
<th># of Inconsistencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects with multiple parents</td>
<td>8</td>
</tr>
<tr>
<td>Parallel edges (typed)</td>
<td>9</td>
</tr>
<tr>
<td>Parallel edges (un-typed)</td>
<td>21</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>38</strong></td>
</tr>
</tbody>
</table>

Table 19. Correct groups of Inconsistency in Hospital

Total number of the inconsistencies in the first step of GCC is 406+38=444. Additionally, the rules ‘Null rule’, ‘All nodes’ and ‘ All edges’ are related to tool debugging rather than to consistency checking. But the rule ‘Un-Navigable’ is related to the structure of model designs. As a result, the rate of ‘False positive’ inconsistencies is 4.05 %, as follows;

\[
\text{Un-Navigable nodes ÷ Total} = \text{rate of False positive} \% \\
18 ÷ 444 = 0.0405 \times 100 = 4.05\% 
\]

4.2.4.4. False Negative

The number of ‘False negative’ errors are not calculated because of not performing resolution approach.

5. DISCUSSION

We conducted two ‘Initial system of models’, “Star” and “One against (N-1) models”, in two problem domains (See Section 4.1 and Section 4.2). The aim for choosing two domains is to operate and validate a more confidence of working on two different systems of models. It means that, having two problem domains, i.e. Warehouse and Hospital, avoid GCC methodology and the results to be affected by getting familiar with a domain, e.g., Warehouse. The results of our research questions have been defined as follows;

RQ1. Does GCC enable finding inconsistencies, which are hard to detect or not detectable at all by using PCC?

According to Case study Execution (Section 4), during all seven steps for consistency checking, we resulted that by GCC, the inconsistencies can be detected for our different models of the problem domain and there can be various inconsistencies over more than two models, i.e. pairs of models. Exploring and refining those groups of inconsistencies which are not detectable or hard to detect by PCC is one of the contributions in this thesis. For instance, as illustrated in Figure 24 (Section 4.1.5), ‘Multiple Parents’ inconsistency is detected by ‘Multiple Inheritance Rule’ between more than two models and the resolution is provided by GCC (See Figure 25). Therefore, GCC enables to detect and resolve such inconsistency which needs more effort by PCC, e.g., in terms of having more comparisons.

According to Table 9 and Table 16, the inconsistencies that were global or local (pair-wise), have been detected by GCC in order to clarify the answer for RQ1.

By this study, we achieved that GCC can help us to diagnose inconsistencies over more than two models and trace back to the origin of the inconsistency in the merged models (See Figure 24 and
Figure 25). Also, there are some inconsistencies that pair-wise checking cannot, whereas, GCC can detect them between more than two models.

**RQ2. Does GCC scale when we have a large number of models and mapping?**

Regarding the scalability of GCC for large number of models, both case studies helped us to find the aspects of scalability. For the first one, we had 31 mapping relationships between the models and for the second one, we had 56 mapping relationships. According to Table 10, GCC scales in different models with e.g., different number of elements (i.e., Model Size). Also, as illustrated in Table 8, we had different number of inconsistencies in each stage of the resolutions. We checked ‘Time duration’ of GCC with increasing number of models and resulted the scalability in approximate linear boost for each consistency checking rule (See Figure 30 to Figure 44). The mentioned tables and figures suggest that, we can scale up to much larger system of models with more elements and relationships. So, we have achieved the scalability of GCC.

**RQ3. Is global-consistency checking feasible to incrementally resolve (elaborate) inconsistencies between DM?**

For answering this research question, we have resolved the inconsistencies through different stages and related strategies for inconsistency elaboration. For each group of inconsistencies, related resolving strategy has been selected and employed. As mentioned in Section 4.1.5, all the stages and procedures have been defined. As a result of these procedures in each stage, the final merged model is consistent. Table 8 shows the number of different inconsistencies and the models’ elements count in each released merge. So, according to Table 8 and the stages of inconsistency resolution, we resulted that, GCC is feasible to incrementally resolve (elaborate) inconsistencies between DM.

**RQ4. How does the elaboration affect the mappings?**

For answering this research question, we focused on the relationships of our models and their changing during resolution approach (Section 4.1.5). According to this section, in order to resolve the inconsistencies, we made a connector between the models. Then, the connector mapped with the models based on the elements defined in the connector, so, the mapping between our models has been improved. Here is the elements count of the connector, which has been made for the inconsistency resolution:

**Element counts:**
- Node count: 54
- Edge count: 52
- Total: 106
Table 20 shows all the number of model’s entity and relationships in the first and last steps of resolution approach.

<table>
<thead>
<tr>
<th>Model #</th>
<th>Entity count</th>
<th>Element count after Refuted</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aggregation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Refuted Aggregation Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Refuted Generalization Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Refuted Link Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Refuted DirectedLink Count</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model #</th>
<th>Entity count</th>
<th>Element count after Refuted</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18 21 39</td>
<td>16 16 32</td>
<td>2 2 8 44 12 0</td>
</tr>
<tr>
<td>2</td>
<td>31 22 53</td>
<td>30 22 52</td>
<td>3 0 7 59 21 0</td>
</tr>
<tr>
<td>3</td>
<td>27 18 45</td>
<td>24 17 41</td>
<td>7 0 5 1 25 3</td>
</tr>
<tr>
<td>4</td>
<td>24 9 33</td>
<td>22 9 31</td>
<td>2 1 0 0 23 3</td>
</tr>
<tr>
<td>5</td>
<td>25 9 34</td>
<td>21 6 27</td>
<td>3 0 5 1 19 5</td>
</tr>
<tr>
<td>6</td>
<td>18 14 32</td>
<td>18 12 30</td>
<td>5 2 3 0 14 2</td>
</tr>
<tr>
<td>7</td>
<td>43 27 70</td>
<td>36 25 61</td>
<td>21 4 19 5</td>
</tr>
<tr>
<td>8</td>
<td>15 8 23</td>
<td>14 8 22</td>
<td>2 1 9 1 8</td>
</tr>
<tr>
<td>9</td>
<td>26 8 34</td>
<td>25 6 31</td>
<td>22 10 0</td>
</tr>
<tr>
<td>10</td>
<td>18 6 24</td>
<td>1 2 21</td>
<td>0 0 0 38</td>
</tr>
<tr>
<td>11</td>
<td>19 4 23</td>
<td>2 0 21</td>
<td>3 1 5 21</td>
</tr>
<tr>
<td>12</td>
<td>17 13 30</td>
<td>0 2 28</td>
<td>0 0 4 32</td>
</tr>
<tr>
<td>13</td>
<td>20 10 30</td>
<td>4 0 26</td>
<td>0 0 2 14</td>
</tr>
<tr>
<td>14</td>
<td>15 22 37</td>
<td>0 1 36</td>
<td>2 0 3 34</td>
</tr>
<tr>
<td>15</td>
<td>33 25 58</td>
<td>2 4 52</td>
<td>13 3 0 63</td>
</tr>
<tr>
<td>16</td>
<td>37 15 52</td>
<td>10 0 42</td>
<td>0 0 4 70</td>
</tr>
</tbody>
</table>

Table 20. Entity-Relationship Count (First and last step)

Table 21 shows the number of correspondences between the connector and the models in step 5 of inconsistency resolution.

<table>
<thead>
<tr>
<th>Model #</th>
<th>Number of correspondences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>36</td>
</tr>
<tr>
<td>14</td>
<td>26</td>
</tr>
<tr>
<td>15</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 21. Element Correspondence of the Connector

Figure 46 shows the evolution of Mapping 1.2 from the original version to the final one.
In this mapping relationship, we can see some new mappings which are defined and some are refuted. In Figure 46, mapping numbers of 55, 67, 68, 70, 71, 127, 128 and 136 are new and mapping 143 is deleted. For all our mappings from Model 1 to all fifteen models, Table 22 shows the changing by the number of new and deleted mappings. These changes are because of defining threshold 0.7 in initial steps of mappings and then, adding and deleting some mappings during *resolution approach*. Table 22 shows the mapping evolution from the first version to the final.

<table>
<thead>
<tr>
<th>Mapping #</th>
<th>Mapping count Final version</th>
<th>Mapping count Final version</th>
<th>Deleted mapping count</th>
<th>New mapping count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>19</td>
<td>26</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1.3</td>
<td>10</td>
<td>14</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>11</td>
<td>17</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.7</td>
<td>12</td>
<td>20</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1.8</td>
<td>5</td>
<td>11</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1.9</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1.10</td>
<td>7</td>
<td>9</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1.11</td>
<td>12</td>
<td>19</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1.12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.13</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.14</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1.15</td>
<td>20</td>
<td>19</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1.16</td>
<td>11</td>
<td>13</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>141</strong></td>
<td><strong>184</strong></td>
<td><strong>12</strong></td>
<td><strong>55</strong></td>
</tr>
</tbody>
</table>

Table 22. Mapping Evolution of Models
So, after all processes of new mappings the quality of mappings is improved. When the elaboration is completed, we made a comparison between initial hypothesizes for the threshold of automatic computed mapping (based on N-gram algorithm with the specified threshold 0.7 [49]). We resulted 55 new mappings and 12 deleted mappings. Therefore, we had substantial fraction of mapping changes of 15%.

\[
\begin{align*}
12 \div 474 &= 0.025 \times 100 = 2.53 \\
55 \div 474 &= 0.116 \times 100 = 11.60
\end{align*}
\]

\[2.53 + 11.60 = 14.13 \sim 15\]

This fraction represents that, though, we had high vocabulary similarities in our different models and we were conservative for the mappings, this is a significant change and it shows how important consistency checking is.

6. VALIDITY THREAT

Here threats to validity are noted and the effort (as required) to minimize the effects is discussed;

6.1. Internal Validity

“Threats to internal validity concern issues that may indicate a casual relationship, although there is none. Factors that impact on the internal validity are how the subjects are selected and divided into different classes, how the subjects are treated and compensated during the experiment, if special events occur during the experiment etc. All these factors can make the experiment show a behavior that is not due to the treatment but due to the disturbing factor”[52].

- **Initial system of models**
  As mentioned in Section 4, we selected two systems (i.e. system of models) for the case studies. This selection of systems can be as a threat for the study. ‘One against (N-1) models’ was selected in Warehouse case study. If we selected e.g., ‘Star’ for Warehouse, we would have less result of inconsistencies e.g., duplicate names. But more correspondence between the models was needed and consequently we needed more efforts for merging models. So, each system of models has its own effects on the study results.
  There is no single way to define a system of models, and one may hypothesize different ways of interconnecting the models. The reason for using two strategies (i.e. system of models) was to build more confidence in that the methodology remains feasible, irrespective of the interconnection strategy chosen, in order to minimize such a threat. For example, if a certain strategy leads to 10 times as much effort than another strategy, then the methodology is too sensitive to the strategy, which may hurt applicability.

6.2. External Validity

“Threats to external validity are conditions that limit the possibility for generalizing the results of the experiment outside the scope. External validity is affected by the experiment design chosen, but also by the objects in the experiment and the subjects chosen”[52].
Our scope of GCC between DM is applied to homogenous models, i.e. models described in the same notation. It is not applicable to heterogeneous models, i.e. models described in different notations. Extending the work to heterogeneous models leads to a challenge of merging models described in different notations. Translating heterogeneous models may cause to discard the structural and visual properties of them (See Section 7.1).

**Overlapping models**

The models have significantly overlapping. In practice, if the context is changed we may not have this amount of overlapping. So, we could have less global inconsistency problems.

Regarding the effectiveness of the resolution done in the study, it can be mentioned that, while the models have low overlaps, then any changing in the models has less possibility to ripple through the whole system.

In better words, the resolution in this case (with significant overlaps) is more complex than the case of having less overlaps. Hence, having this amount of overlaps was one of the most useful cases.

**Existence of an upfront problem description**

In these two case studies, problem domains’ descriptions were available. One of the factors out of treatment was the way, in which stakeholders model the problem domain. This factor affects the results and makes different conflicts which force many inconsistencies to be resolved.

It cannot ascertain if GCC remains as useful in situations where the models have less overlaps. But such a degree of overlap is not uncommon in a viewpoints-based paradigm [4]. We believe that no significant reduction in usefulness will be seen if viewpoints are used as the main vehicle for requirements elicitation.

### 6.3. Construct Validity

“Threats to construct validity refer to the extent to which the experiment setting actually reflects the construct under study. It is concerned with the relation between theory and observation”[52].

**Scalability**

We checked the scalability based on models size, as a limited notion of scalability. There are some other aspects for scalability, i.e. number of relationships. In case of having huge number of relationships, navigating of inconsistencies becomes complicated. Also, in case of having huge number of complicated models, the layout of the merged models will affect the scalability due to having a convoluted layout.

Scalability is human centered and other aspects need ‘controlled experiment’ and more detail treatment, which is out of the focus of an exploratory case study (this thesis). It should be mentioned that, we just conducted preliminary study on GCC. So, more consideration of scalability is unrealistic for this thesis. It requires follow-up study with more human centered and controlled experiments, which are seen as a direction for future work.

**Learning effect**

The first aspect is about learning ‘the system of models’ that is addressed in Section 6.1, as the internal validity. The second aspect is related to ‘inconsistency resolution’. The effect of learning on ‘how to resolve inconsistencies’ is notable. In initial rounds of inconsistency resolution, a new user may need more effort to work in this approach. In
continue, when new users get familiar with the domain, the effort would be more proportional with models size.

Mitigating learning effects require a randomized experiment with a sufficiently large number of subjects. Our exploratory case study in this thesis should be viewed as a preliminary step for such experiments. That said, the learning effect does not compromise the results of our experiment, as it only concerns the increased skills of the designers in resolving inconsistencies.

7. CONCLUSION

The research questions have been answered, in particular, through conducting two different case studies, as follows;

For answering RQ1, through our case studies, we concluded there are some types of inconsistency, which are hard to detect and diagnose, in comparison with pair-wise checking, due to rooting from more than two models. The reasons for that generally were the number of model comparisons, inability of PCC for identifying some kinds of inconsistency, etc. GCC enables to provide a whole view (a merged model) of DM, study their effects on each other regarding their relationships, and check it against the consistency rules.

To address RQ2, through our first case study (Section 4.1), we concluded that GCC scales different models with e.g., different numbers of element (i.e., Model Size). It is shown by ‘Time duration’ of applying GCC against each consistency rule (See in Section 4.1.5). ‘Time duration’ diagram has a linear boost with increasing number of models (See Figure 30 to Figure 44). Meanwhile, the rule application effort and inconsistency identification, plus resolving them, could be generalized to different numbers of model (regardless of their size).

For answering RQ3, through the first case study, according to our methodology for GCC (See Section 3 and Figure 6), we concluded that a group of inconsistencies can be resolved in each iteration (from ‘Computing merge’ to ‘Elaboration’ phase), and finally a consistent model will be provided. Table 8 indicates how the inconsistencies were resolved by resolution approach (See Section 4.1.4), when we assure the clearance of previous group of inconsistencies. Therefore, GCC is feasible to resolve inconsistencies of DM incrementally.

To answer RQ4, we studied the changes that were made to the mappings during inconsistency resolution (see Table 21). These changes include the establishment of new mappings that did not exist in the initial relationships computed by the matcher as well as the removal of some of the mappings in these initial relationships. In our case study, from a total of 474 element mappings in the initial relationships, 12 ones were refuted (deleted) and thus removed during resolution process. In addition to these removals, a total of 55 element mappings were introduced. These indicate a difference of 15% in the mappings when the initial relationships are compared to the final ones.

Using GCC facilitates tracing inconsistencies between models back to root-causes and resolving them for a project of DM. Our effort in this thesis was to focus on the need of applying model management activities, in particular, consistency checking over systems of models (i.e., globally), rather than over individual models and/or pairs of models. We tried to indicate such an approach can potentially address many problems occurred in distributed software projects. As a whole, the contribution of the thesis is to study how the provided methodology in [1] is applicable and feasible to explore and refine
the inconsistencies between distributed UML class diagram models. In fact, the aim was to conduct a preliminary study on checking consistency via merge and match between more than two models and analyze the results in order to answer our research questions.

7.1. Direction for Future Works

This thesis has some shortcomings that we aim to address in the future. GCC requires follow-up study with more human centered and controlled experiments in order to be generalized, in terms of scalability, learn-ability and improving usability. The next fundamental direction for future works is to expand GCC for ‘heterogeneous’ models. Our study works only on ‘homogeneous’ models because ‘merge’ cannot be applied to a notational level. In future work, we plan to extend and provide our GCC methodology to ‘heterogeneous’ models. A challenge for heterogeneous consistency checking would be involve merging models represented in different notations. One solution could be merging a set of heterogeneous models by translating them into a single notation first, but such a translation discards the structural and visual properties of the models [1, 3, 5 and 21].
REFERENCES


APPENDIX

Appendix A

1. Warehouse

A warehousing company has a system to handle orders for minivan bumper fascia of various colors from an automotive factory. (Fascias are covers for bumpers.) The system tracks the status of the orders, provides computer support for warehousing workers, and keeps track of inventory levels in the warehouse. Workers pick fascia off the warehouse floor and load individual fascia into pallets, and load the pallets into trucks. If a truck arrives at the automotive factory with even one fascia in the incorrect order in the truckload, the warehousing company is fined tens of thousands of dollars because such an error holds up the entire assembly line.

Warehouses are organized into zones, and the zones have aisles of shelves. A pick face in a warehouse is one side of a shelving unit that holds racks of products (in this case, fascia). The "floor" of the warehouse is the area in which the warehouse pickers are working and the "reserve room" is where the bulk of the inventory is stored.

The warehouse handles the selection and loading of fascia for bumpers on minivans. They sort and deliver pallets of fascia to the automotive factory for direct loading into their assembly line.

Each order from the automotive company describes a single minivan, including color and model. The software will translate that into a pair of fascia for the minivan, front and back, which must be sequenced in the right order on the pallets so that when they arrive at the factory and are loaded into the assembly line, each pair of fascia is matched with the right color minivan.

Pickers are on the floor of the warehouse. They drive forklifts from rack to rack, picking fascia of various colors from pallets, and taking them to a marshalling area for sequencing. They are then placed on a special pallet designed to protect them as they are shipped 4 fascias per pallet. Pickers have a handheld device with a barcode reader that will direct them to the next zone, aisle, rack location and level on the rack containing the next fascia to pick, and the barcode reader will read the SKU of the fascia they just picked. After sequencing, the SKU is registered again to mark that it has been sequenced.

In order to minimize picking time, there is already generic software to tell pickers the order in which to traverse the warehouse with their forklifts, because an un-optimized traversal wastes a time (and money).

Each truck holds 40 orders (80 bumpers) that are stacked 10 high. Each front/back pair of pallets needs to be placed in the right order on the truck, facing the right direction. The pallets are loaded in pairs, front bumpers and matched back bumpers, so that there is no mismatched or missing fascia at the automotive factory.

When a new shipment of fascia arrives, it is unloaded from the truck and checked for problems such as damaged fascia or incorrect colors. It is then entered in the system and put away in the warehouse reserve room, which is where racks on the warehouse floor are resupplied from. (There is already code that optimizes the put-away process.)

When a pick face gets low, it triggers a replenish request to get more fascia of the type that is running low from the reserve room. When replenishing happens, the re-supplier records that information so that the system knows that the fascia have been moved to the pick face.
1.1. Order:

An order is for a single minivan, including the model and color of the minivan. Orders will be received by a single FAX machine in the warehouse, and kept in the order in which they are received.

Using a translation table, you look up SKU numbers for front and back fascia. The translation table looks like this:

<table>
<thead>
<tr>
<th>Colour</th>
<th>Model</th>
<th>SKU (front)</th>
<th>SKU (back)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>XLE</td>
<td>11203</td>
<td>10432</td>
</tr>
<tr>
<td>Yellow</td>
<td>LE</td>
<td>33104</td>
<td>42105</td>
</tr>
<tr>
<td>Maroon</td>
<td>CE</td>
<td>21444</td>
<td>21576</td>
</tr>
</tbody>
</table>

Four orders will be processed at a time. Until there are four, they should not be entered in the system. Once there are four orders, the colors and models of the four minivans are entered into the system. The system then looks up the SKUs for the front and back fascia and produces a picking request for the four pairs of fascia. Each picking request receives a unique id.

1.2. Picking:

Pickers are on the floor of the warehouse. They drive forklifts from rack to rack, picking fascia of various colors from pallets, and taking them to a marshalling area for sequencing. They are then placed on a special pallet designed to protect them as they are shipped 4 fascias per pallet. Pickers each have a handheld barcode reader that has some sort of interactive display and a wireless connection to the software system.

When a picker is ready, they will ask the system for the next picking request to process. The barcode reader will instruct the picker to get eight fascias, one at a time, for the four minivans in the picking request.

On the warehouse floor, each rack on a pick face holds a single kind (color and model) of fascia.

The barcode reader directs them to the next zone (identified by a capital letter), aisle (identified by an integer), rack location in the aisle (an integer), and level on the rack containing the next fascia to pick (again, an integer). After placing a fascia on the forklift, the picker uses the barcode reader to record the SKU number of that fascia. When all eight fascias have been gathered on the forklift, the system directs the picker to the marshalling area for sequencing.

In order to minimize picking time, there is already generic software to tell pickers the order in which to traverse the warehouse with their forklifts, because an un-optimized traversal wastes a time (and money). This software has a Java wrapper available in a Java jar file. This generic software is kept up to date by a separate group in the warehousing company, and they have entered the locations of all the kinds of fascia in the warehouse, and the new developing system needs merely to ask for the 8 locations on the warehouse floor by sending the generic software the a list of 8 SKU numbers; it will return the list of SKU/location pairs in the correct order to be picked.

1.3. Sequencing:

The eight fascias must be placed in two special pallets. One pallet will contain the four front fascias, and the other pallet the four rear fascias, in the same order that the minivan orders came in. This process is called sequencing.
After the sequencer sequences the fascia onto the pallets they visually inspect that they have the fascia in the correct places. They then record all eight SKUs using a barcode reader, first the front fascia and then the back fascia in the same order they were sequenced, to mark that they have been sequenced and to verify that they are in the correct order. If any are incorrect, it must be because the picker picked a wrong fascia. The current set of fascias is discarded, and 8 new ones must be retrieved.

1.4. Loading:

Each truck holds 80 orders (160 bumpers) that are stacked 10 high. Each front/back pair of pallets needs to be placed in the correct order on the truck, facing the right direction. (It is easy to tell which way they should face.) There are four pallets per level.

Loaders will look at the picking request id and, using the barcode reader, scan the SKUs of the fascia to be loaded to make sure that orders are loaded in the correct order, and they will record that the picking request has been loaded. If the next picking request has not yet been processed (for example, a forklift broke down and other pickers finished first) then nothing will be loaded until the picking request arrives.

Trucks never leave the warehouse unless they are fully loaded.

1.5. Supply:

When a new shipment of fascia arrives, it is unloaded from the truck and checked for problems such as damaged fascia or incorrect colors. In case of any problem, fascias are not entered in the system.

Fascia come in crates of 40 (all of them front fascia or all of them back fascia, never mixed), all for the same model and color of minivan. They are entered into the system and put away in the warehouse reserve room, which is where racks on the warehouse floor are resupplied from. There is already code that optimizes the put away process; it is a separate system and the new developing system does not need to interact with it.

1.6. Replenishing:

Whenever a picker records that a fascia has been picked, the system checks whether that kind of fascia is running out on the level of the rack it is on. This happens when there are exactly 5 of that kind of fascia left. If there are 5 left after picking, the system triggers a replenish request to get 25 more fascia of the appropriate type from the reserve room. When replenishing happens, using the barcode reader the re-supplier records the SKU for the fascia so that the system knows that more fascia have moved from the reserve room to the pick face.

By the nature of picking and replenishing, the pick faces will never run out of any kind of fascia.

The reserve room never runs out of fascia, if the warehouse does not have enough fascia of a particular color and model, the automotive company will not send orders for minivans of that color and model.
2. Hospital

The hospital gives around-the-clock medical care, diagnosis and treatment to the sick and injured on both an inpatient and an outpatient basis. The hospital is divided into several wards, described below. The medical team is comprised of physicians, technicians (technical staff), nurses, and administrative assistants.

2.1. Wards, rooms, units, beds and equipment

Wards have rooms for patients to stay in. Rooms have beds, and beds are numbered per room, so a combination of the ward, room, and bed number will uniquely identify every bed in the hospital.

The pieces of equipment in a ward can be mobile or stationary. If stationary, a piece will be assigned to a single ward room or unit; and if mobile, it will be assigned to the general storage area in the ward. Equipment is not shared between wards. For each piece of mobile equipment, there is a booking calendar to keep track of when and where the piece is needed.

Wards also have units, such as critical care units and operating room units. The set of units varies from ward to ward, but every ward has a single general storage area unit. Each unit has a textual description outlining its responsibilities, as well as a list of equipment in that unit.

2.2. Display units and scanners

Every ward unit and room has a display and a scanner. The display can show everything from a medical team member's schedule to a patient's history. The scanner is connected to the display, and can be used to scan both the wrist band of a patient (described below) and the id card of a member of the medical team.

2.3. Medical team

Each medical team member has a name and a unique id number, and all team members carry an id card containing that information.

Nurses are affiliated with a single ward, while physicians and technicians can be affiliated with several different wards. All personnel have access to a calendar detailing the hours that they need to be present at the various wards. Nurses record physicians’ decisions such as diagnoses, medical procedures that are to be performed (and when), new prescriptions (including medication, amount, and schedule), cancelled prescriptions, whether a patient needs to be transferred (and to which ward), and whether a patient can check out of the hospital. These are written on paper and handed to an administrative assistant to enter. The administrative assistant needs to figure out who needs to be at a particular procedure before they enter it in the system.

Here are some necessary items for the system:

- Each procedure has a list of medical equipment necessary to perform that procedure. A procedure will only be scheduled when all pieces of equipment are available.
- The technicians can view their schedule and move equipment as necessary.
- Physicians have a set of patients they are assigned to. Nurses and physicians can review a patient's medical history.
• Everyone needs to be able to see a list of the medical procedures in which they are involved.
• Physicians need to be able to get a list of their patients to be visited in a ward.

2.4. Patients

Unless new patients are in a life-threatening situation, they must register at the hospital. This involves an administrative assistant scanning their health card, which contains their name, address, and a unique health card number. After registering they receive an early diagnosis by a physician in which they are classified as either an "inpatient" or an "outpatient". An inpatient is a person who is admitted at least for one night to the hospital; and an outpatient is a person who visits the hospital for diagnosis or treatment without spending the night. This is determined by the physician doing the initial assessment.

Outpatients on a return visit may arrive at the hospital and go straight to the ward where they are to receive treatment.

Each patient, be that an inpatient or an outpatient, has a profile capturing the patient's name, address, health card number, and their general health-related remarks. For each outpatient visit, an outpatient visit record is created storing the date and time of the visit, the patient's health problem, the name(s) of the physician(s) who attended to the patient, the diagnosis, and the prescribed medication. For each inpatient visit, an inpatient visit record is created storing the date and time of admission, the patient's health problem(s), the early diagnosis, and the name(s) of the physician(s) involved in making the early diagnosis. To facilitate the management of inpatients, a wristband is produced for each inpatient at admission. The hospital's medical team will use special handheld scanners to scan the wristbands and fetch the inpatient visit records, described below.

During each stay at the hospital, an inpatient may be transferred several times between different wards and between different parts of a single ward. For each ward stay, a ward stay record is created. The information stored in a ward stay record includes a list of intra-ward stays, a list of medical procedures performed on the patient during their stay at the ward, the name(s) of the physicians attending to the patient while at the ward, a ward-specific diagnosis and a ward-specific medication chart. The ward nurses use these charts to administer the required medication. The information stored in an intra-ward stay record includes the date and time when the patient checked in the corresponding ward room or unit, the checkout date and time, and if applicable, the id of the bed assigned to the patient within the room or unit.

Patients in a life-threatening situation are assessed and treated as normal, but without registering them first. They are all issued a wristband at some point for identification, and flagged in the system as not having registered.
Appendix B

Raw Models

All raw models, which were developed by senior students (mostly graduated) [12], are illustrated in this section. The raw models are in class diagram notations and elicited from different stakeholders. There are sixteen viewpoints for Warehouse (Vw1 to Vw16) and eight ones for Hospital (Vh1 to Vh8).

1. Warehouse

Vw1.

Vw1.1.
Vw1.2.

Diagram:

1. Team uses 1 Terminal Device
   - ID
   - State
   - Set UI
   - Get Input from Keyboard

2. Desktop Computer
   - 1..* to Barcode Reader
     - Get Input from Screen
     - Scan SKU Number

3. SKU Translation Table
   - Colour
   - Model
   - Front Fascia SKU
   - Rear Fascia SKU

4. Desktop Computer
   - reads SKU Translation Table to check the orders

5. Fascia

Arrow labels:
- uses
- 1..*
- reads
- describes
Vw1.3

There is 1 Picking Request Queue per Truck, so that the system can process multiple trucks in parallel.
Vw2.

Vw2.1
Vw3.

Vw3.1

Structure of the Warehouse Company and its Employees

[Diagram of the warehouse company structure with labels and connections between different areas and departments such as Picker, Sequencer, Loader, Replenisher, Administrative Assistant, barcode reader, marshalling area, loading bay, reserve room, floor, warehouse company, and office.]

Only one Administrative Assistant?

Mostly works in Reserve Room, but needs to go to the floor as well.
Vw4.

Vw4.1

**INCOMING ORDERS**

![Diagram of INCOMING ORDERS](image)

Each Picking Request produced is associated to 4 pairs of fascia.

Vw4.2

**PICKING**

![Diagram of PICKING](image)

Each forklift is to get 8 fascia in total; one at a time.

The traversal program returns 8 locations at a time.

Scan multiple fascia’s SKU; 8 scans per unload.

This is done using the already existing traversal program.
1.3 SEQUENCING

Picker and Sequencer communicates with each other to ensure sequencing of the current Picking Request was successful.

2 Sequencers share a single barcode reader.

It is possible for Fascia to be discarded during sequencing.

Fascia unloaded to the Marshalling Area get sequenced to Pallets.
1.4 LOADING

Loaders share a single barcode reader.

1.5 SUPPLY AND REPLENISHING

Each new shipment contains 40 fascia of the same type.

Resuppliers access the reserve room to replenish fascia.

After Picker records that a fascia has been picked, there may be a resupply request; in which case, the barcode reader will provide the resuppliers a notification.
Orders
Vw5.3

Sequencing

Vw5.4

Loading
Vw5.5

Supply

- Shipment
  - contains 1
  - *

- Truck
  - contains 1

- Crate
  - front: bool
  - for_model: string
  - color: string

- Resupplier
  - operate
  - resupplies and scans

- HandHeldBarcodeReader
  - + scanSKU()
Replenishing

- WareHouse
- Floor
- Reserve Room
- Rack
  - containsFasicaOfColor : string
  - containsFasicaOfModel : string
  - quantity : int
- Resupplier
  - HandHeldBarcodeReader
    - operate
      + scanSKU()
- Replenish Request
  - [quantity = 5]
    - Triggers
- Fascia

Employees

- WareHouse
- Employee
  - Administrator
  - Loader
  - Resupplier
  - Sequencer

work at
Vw6
Vw6.1

Orders

Vw6.2

Warehouse Organization
Vw7

Vw7.1

Workers

Vw7.2

Units and Rooms
Equipment, Transportation, Storage
Orders, Fascia and Tracking
Vw8.

Vw8.1

**Warehousing Personnel**

![Diagram of Warehousing Personnel]

Vw8.2

**Warehouse Layout**

![Diagram of Warehouse Layout]
**Warehouse Software Interfaces**

![Diagram showing the interface relationships between Order GUI, Barcode Reader GUI, and TraversalSoftwareInterface.](image-url)
Workers

WarehouseWorkers
- name : string
- id : int

Administrator

Sequencer

Picker

Loader

Resupplier

TranslationTable

PickingRequest

PickUpQueue

BarcodeReader

SequenceQueue

SpecialPalletQueue

FrontFascia

RearFascia
Orders

Vw10.

Vw10.1
Supply

- **Resupplier**: 2..* (unloading)
  - Has a 1
    - **Warehouse**: Has 1
      - **System**: 1 (translation table, lookup SKU num)
        - Connected to 0..* (Handheld Barcode Reader)
          - 0..1

- **Truck**: 1..* (has)
  - 40..* (Fascia)
    - **Fascia**: SKU num
2.6 Replenishing

[Diagram showing relationships between System, Warehouse, Picker, Replenisher, Handheld Barcode Reader, and Fascia.]
Class Diagram for Order and Picking Request creation.
Vw12.2

Class Diagram showing relationship among warehouse workers.

Vw12.3
Vw13.

Vw13.1

Orders

Vw13.2

Picking
Vw13.3

**Sequencing**

[Diagram showing the process of sequencing]

Vw13.4

**Loading**

[Diagram showing the process of loading]
Vw13.5

Replenishing

![Diagram of Replenishing process]

- Barcode Reader: ask for request, record SKU
- Picker: replenish request
- Replenishing Team: record SKU
- Fascia: replenishes supply
- Location: zone, aisle, rack, level, stock

110
Vw16.

Vw16.1

Order
Picker

Vw16.2
Vw16.3

Sequencer

Vw16.4

Loader
2. Hospital

Vh1.

Vh1.1
Vh1.5
Vh2.

Vh2.1

Wards, rooms, units, beds and equipment

Vh2.2

Medical team
Vh2.3

Patients

Physicians  Assignment

Visit Record

inVisit record  outVisit record

In-patient  Out-patient

Medical procedure

Vh2.4

The whole high-level class diagram

Technicians  Nurses

Schedule

Medication

Assistants  Physicians

Visit Record

prescription

inVisit record  outVisit record

Out-patient  In-patient

WardStay-record

Medical procedure

Wristband
Vh3.
Vh3.1

Wards, rooms, units, beds and equipment
Display units and scanners

- Room
- Unit
- Schedule
- Patient
- Booking calendar
- Calendar
- Scanner
- Display
- Wristband
- ID Card

Relationships:
- Room has 1 Unit
- Unit has 1 Schedule
- Unit has 1 Patient
- Unit has 1 Booking calendar
- Unit has 1 Calendar
- Scanner connects to Display
- Scanner scans 1 Wristband, scans 1 ID Card
- Display 1.0 accesses 1 Scanner
- Display 1.0 shows 1.0 Wristband
- Display 1.0 shows 1.0 ID Card
- Display 1.0 shows 1.0 Schedule
Vh5.

Vh5.1

High-level class diagram.

Vh5.2

Patient class
Vh5.3

MedicalTeam class

Vh5.4

Equipments class

Vh5.5

Calendar diagram
Vh6.
Vh7.

Vh7.1

Wards, rooms, units, beds, and equipment
Vh7.5

Records associated with wards

- **Physicians**
  - Patient profile
    - name: String
    - address: String
    - health card #: Int
    - gender: String
    - birth date: Date
  - Ward
    - ward ID: String
  - prescription
    - medication: String
    - amount: Int
    - schedule: String
    - active: boolean
    - action: String
    - action
  - medical chart
  - diagnosis
    - description: String
  - medical procedures
    - description: String
    - ID
  - intra-ward stay record
    - check-in date: Date
    - check-out date: Date
    - check-in time: Time
    - check-out time: Time
    - room: String
    - ID: Int
    - note: String

- Records associated with wards
  - has a ward specific
  - contains a ward specific
  - performed at
  - contains a ward specific