Automated Testing of Object Oriented Systems using VDM++ and UML Communication Diagrams

A Dissertation Submitted by

AAMER NADEEM

PC 033004

Towards the Degree of

Doctor of Philosophy (Computer Science)

Mohammad Ali Jinnah University

January, 2007

Automated Testing of Object Oriented Systems using VDM++ and UML Communication Diagrams

A Dissertation Submitted to

MOHAMMAD ALI JINNAH UNIVERSITY

Towards the Degree of Doctor of Philosophy in Computer Science

BY

AAMER NADEEM

PC 033004

Supervisor: **Dr. Muhammad Jaffar-ur-Rehman (Late)**
Professor, Mohammad Ali Jinnah University
Islamabad, Pakistan

Co-supervisor: **Dr. Michael R. Lyu**
Professor, The Chinese University of Hong Kong
Hong Kong S.A.R., China

Mohammad Ali Jinnah University

January, 2007

Dedicated to,

The Memory of Late Dr. Muhammad Jaffar-ur-Rehman and his Family

Abstract

The rapidly growing applications of software in critical systems such as railways, aviation, automobiles, and medicine, demand a much higher level of reliability and error-free operation. The use of formal methods in such applications not only helps avoid specification errors, ambiguities, and inconsistencies in early phases of software life cycle, but also provides a sound basis for generation of an effective set of test cases. However, the existing research on formal specification based testing has focused on unit level testing only.

This research is aimed at automating the generation of class level as well as integration level test cases for an object-oriented system using formal specifications. We use VDM++ formal specification language for this purpose. As a result of our research, we present a framework, called *SpecTGS*, that automatically generates specification based test cases for object-oriented systems using VDM++ as the specification language. For class testing, the *SpecTGS* uses the trace structure definition of a VDM++ class specification to derive allowable method call sequences, and partition analysis to generate test data. For integration testing, we have proposed a novel idea that extracts testing information from the VDM++ specification and UML communication diagrams. The *SpecTGS* derives message sequences from a UML communication diagram, and uses the VDM++ specification to construct state invariants for the states in which a class can receive a message. A new strategy for constructing sub-states from a state invariant called *partitioned boundary state coverage* that combines two existing strategies, i.e. partition

analysis strategy and the boundary state coverage strategy. Each message sequence generated from the UML communication diagrams is combined with the sub-states to construct a test model. The test model is then used to derive the test paths under various coverage criteria. A proof-of-concept tool has been developed to implement and evaluate the *SpecTGS* framework. The results for the integration testing approach have been shown for a real-life case study selected from the literature.

Acknowledgements

First of all, I thank the Almighty Allah for giving me the strength and determination to undertake and complete this enormous task. Without His blessings, it would have been impossible for me to achieve this goal.

I express my deepest gratitude to my supervisor Professor Dr. Muhammad Jaffar-ur-Rehman (late), Former Dean, Faculty of Engineering and Sciences at Mohammad Ali Jinnah University (MAJU), for sparing time from his extremely busy schedule, whenever I needed his guidance. His supervision and advice played a key role in successful completion of this work. Unfortunately, he passed away in the devastating earthquake that hit northern Pakistan in October 2005.

I am deeply indebted to Professor Dr. Michael R. Lyu, Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK), for his valuable guidance, support, and useful discussions on my work during my three-month visit to the CUHK in summer 2005. Not only did he help and guide me during my stay at the CUHK, but also he was kind enough to act as my co-supervisor after sudden and unexpected death of my supervisor. I have no hesitation in acknowledging that it was his support that helped me complete my work in a timely manner.

I am strongly indebted to the management of Mohammad Ali Jinnah University for their full support and cooperation which enabled me to continue and complete my work even

after I lost my supervisor. In particular, I would like to thank Mr. Syed Ali Imran, Executive Vice President, Professor Dr. Muhammad Mansoor Ahmad, Dean Faculty of Engineering and Sciences, and Professor Zafar I. Malik, Head of the Computer Science Department.

I am also thankful to the foreign experts who evaluated my thesis, namely Eric Wong, Associate Professor, University of Texas at Dallas (USA), Jin-Song Dong, Associate Professor, National University of Singapore (Singapore), S.C. Cheung, Associate Professor, The Hong Kong University of Science and Technology (Hong Kong S.A.R., China), and ZhiQuan Zhou, Lecturer, University of Wollongong (Australia). They offered valuable comments which helped me improve my work.

I thank my colleagues and students at the Center for Software Dependability (CSD), Mohammad Ali Jinnah University, for bearing with me during this difficult period and for their moral support.

Last but not the least, I express sincere thanks to my family and friends, particularly my wife for her moral support, her prayers, and her patience.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Safety-critical systems are rapidly becoming commonplace in our lives [KT98]. The human society now has greater-than-ever dependence on software or software-controlled systems, and this dependence is growing day-by-day. Software-controlled systems have found their way into almost all walks of human lives, such as aerospace, aviation, defense, nuclear power plants, industrial robotics, medicine, automobiles, railways, and home appliances. Littlewood and Strigini [LS00] identify the various dimensions of the society's dependence on computer systems, as follows:

a) Software-based systems are replacing older technologies in safety- or mission-critical applications, such as aircraft engine control, railroad interlocking, and nuclear power plant protection.
b) Software is moving from an auxiliary to a primary role in providing critical services, e.g., in air traffic control systems.
c) Software is becoming the only way of performing some function which is not viewed as *critical* but whose failures would deeply affect individuals or groups,

e.g., databases and software used by hospitals, super markets, airline reservation etc.

d) Software-provided services are becoming increasingly an accepted part of everyday life without any special scrutiny, such as the widespread use of spreadsheet programs in decision-making.

e) Software-based systems are increasingly integrated and interacting, often without effective human control. For instance, in large, closely coupled systems the effects of software failures can propagate more quickly, and with little or no human control.

The controlling software at the core of such systems is of critical importance, since its failure can result in a catastrophe, loss of human life, significant financial loss, or damage to the environment. Some well-known disasters that occurred due to failure of a critical software include the ARIANE 5 rocket launch disaster [Lio96], the Therac-25 radiation therapy machine failure [LT93], and the London Ambulance Service accident [FD96]. Thus, safety and reliability of such software systems are of paramount importance.

A major source of software faults is an erroneous, inconsistent, or ambiguous specification. A reliable and fault-free software cannot be produced from an ambiguous specification. Since natural languages are inherently ambiguous, it is practically impossible to write a precise and unambiguous software specification in a natural language. Semi-formal notations such as the Unified Modeling Language (UML) are

widely used in the industry for modeling object-oriented systems, but they lack formal semantics.

Formal methods are widely recognized as a means to write precise, consistent, and unambiguous specifications [KT98]. Bowen and Hinchey point out that formal specification techniques make the specifications more concise and explicit [BH95]. Clark and Wing conclude that interest in using formal methods is growing because of their potential to improve software quality [CW96]. The use of formal methods helps avoid specification errors, ambiguities, and misinterpretations in early phases of software life cycle. Unlike natural languages, formal languages are based on sound mathematical principles, and allow aspects of the specification to be rigorously demonstrated using mathematical proofs. However, the use of formal specification methods, by itself, provides no guarantee that the implementation will be correct, or it will conform to the specifications.

The widely recognized role of formal methods in program verification is their use in a correctness proof. A formal proof of correctness is usually done by proving program properties, since it is impossible to prove the total correctness of an arbitrary program due to the undecidability of the halting problem. Moreover, a formal proof of correctness is not cost-effective, or even practical, for most software systems because of the complexity and size of any non-trivial software system. Even after a formal proof, testing is required to build confidence in the system being developed [Meu98]. Therefore, the need for

rigorous testing is not eliminated by the use of formal methods. In fact, formal methods and testing are complementary to each other.

However, even for the most trivial systems, exhaustive testing is practically impossible due to the resource constraints and almost infinite combinations of input values to be tested. Thus, it becomes necessary to find ways to identify a representative set of test cases. For large and complex systems, manually generating such a set of test cases, executing them, and comparing the results with expected outputs can be a tedious and time-consuming process. Fortunately, the existence of a formal specification provides an opportunity to automate much of the testing process, from test case generation to test execution to results evaluation. The model-based formal specification languages such as Z, VDM, and their object-oriented dialects (Object-Z, Z++, and VDM++) have been used to automate the generation of test cases, e.g., [DF93] [SC96] [Meu98] [LMZ02].

In this dissertation, we focus on a particular formal specification language, i.e. VDM++, to automate the generation of specification based test cases. We develop a complete test generation framework, called *SpecTGS,* for object-oriented systems, that generates both unit and integration level test cases. The *SpecTGS* uses a VDM++ specification to generate test cases for class testing, as well as inheritance and polymorphic testing. However, generation of integration level test cases requires that the specification define the dynamic interactions between the objects. An implicit VDM++ specification (as well as other model-based formal notations such as Object-Z) lacks this information, i.e., it does not specify the dynamic behavior of a system – it only specifies the static structure

of the system, i.e. classes, their attributes and operations, and relationships among the classes, which makes it impossible to generate integration tests from the formal specification without supplementing it with some other artifact. As pointed out by Offutt et al., most of the research on formal specification based testing focuses on unit testing only [Off98] [OLAA03]. A major reason for this is that the formal notations are not adapted to specifying the dynamic interactions between the objects. Thus, integration testing of an object-oriented system from a formal specification is still a largely unexplored area, while a lot of the existing work focuses on unit testing.

Thus, for generating integration level test cases, we need this additional information about object interactions, along with the VDM++ specification. For this purpose, we introduce a novel idea that extracts information from two different artifacts, i.e. VDM++ specification and UML communication diagrams, to generate test data. The *SpecTGS* framework uses UML communication diagrams to extract the information about object interactions. The use of UML, together with a formal specification, to specify the system behavior is not a new idea. As observed by Agerholm and Schafer [AS99], as well as other researchers [ELCKAH98] [HCKKTFSMSCM95], despite a lack of formal semantics, the role of the UML is recognized in the formal methods community in decomposing complex problems and presenting abstract visual perspectives of the models. The use of the UML along with a formal specification offers complementary benefits such as visualization of the models, and providing higher-level structural views of the system, while the formal notations can fill in the processing details with their precise and unambiguous syntax [AS99].

The *SpecTGS* framework uses the trace structure definition of a VDM++ class specification to derive allowable method call sequences of a class, and partition analysis together with boundary value analysis to generate test data. For integration testing, the *SpecTGS* derives message sequences from a UML communication diagram, and uses the VDM++ specification to construct state invariants for the states in which a class can receive a message. The state invariants are used to construct sub-state invariants under a novel strategy called *partitioned boundary state coverage* which is based on a combination of two existing strategies, i.e. partition analysis and boundary state coverage. Each message sequence is then combined with the sub-state invariants to construct a test model. The test model is then used to derive the test paths under various coverage criteria. Tool support has also been provided to implement the *SpecTGS* framework, and the integration testing approach has been demonstrated on a case study.

The rest of this dissertation is organized as follows: Chapter 2 gives a brief background of formal specification based testing, VDM++ notation, and the Unified Modeling Language. Chapter 3 presents a literature survey of the existing formal specification based testing techniques, while chapter 4 covers the survey of integration testing techniques based on the UML. Chapters 5 and 6 give the details of the *SpecTGS* test generation framework for unit testing and integration testing, respectively. Chapter 7 covers implementation and a detailed case study that demonstrates the integration testing approach adopted in *SpecTGS*. Finally chapter 8 concludes the work.

Chapter 2

Background

This chapter gives a brief background of the key concepts necessary to the understanding of the ideas presented in this dissertation. In particular, we cover the basic formal specification based testing strategies, an introduction to the VDM++ specification language, the Unified Modeling Language (UML) and its role in testing.

Dijkstra observed that a major limitation of software testing is that it can only show the presence of faults, and not their absence [DDH72]. Despite this obvious limitation, software testing is recognized as a necessary means of program verification. Even when other program verification techniques such as static analyses and formal proofs are employed, testing is still considered necessary to complement these techniques, and to build greater confidence in the system being developed. There are two fundamental approaches to testing, i.e. specification based testing and code based testing. Other approaches use software design models, and combinations of the various artifacts. While code of a software contains detailed information that may be required to generate test cases, a major drawback of testing based on only the code is that the code does not specify what the system *should* do. For instance, if the system implements a wrong

(undesired) function correctly, the tester might never know what the system was supposed to do, if testing is based on code only. Offutt et al. [OLAA03] identify several other advantages of specification based testing over code based testing, i.e.

- test cases can be created before software is coded which, in turn, leads to shifting of testing activities to early phases of software development life cycle and better planning and allocation of resources.
- test generation from the specification helps in identifying inconsistencies and ambiguities in the specification which prevents propagation of specification errors to later phases of software development.
- the essential part of the test data is independent of any particular implementation which enables the test cases to be run on any implementation regardless of the design chosen and the implementation language chosen.

In practice, however, a combination of specification based and code based testing strategies is used to build greater confidence in the system under test. The most common way of doing this is to generate test cases from the specifications and then employ code coverage analysis to assess effectiveness of the generated test suite [OLAA03]. In this dissertation, our focus is on the specification based testing only. We use a VDM++ formal specification and UML communication diagrams to generate test cases. The next section introduces common specification based testing strategies, while VDM++ and UML are briefly described in the last two sections of this chapter.

2.1 Specification Based Testing

Specification based testing, also commonly known as black box testing [Bei95], is a general approach to software testing in which testing information is extracted from the software requirements specification. The software requirements may be described informally in a natural language, or may be modeled using some graphical notation, or may be formally specified in a formal specification language. The advantages of using a formal notation are obvious: ambiguities and inconsistencies can be avoided at the specification stage of SDLC (Software Development Life Cycle), and thus propagation of specification errors to the design and implementation phases can be avoided. Software testing consists of three main activities, i.e., generation of test cases, execution of test cases, and evaluation of results. The goal in the first of these activities is to generate an effective set of test cases that has the ability to uncover maximum faults. Effectiveness of a test case is measured by the number of faults that it can potentially detect. If two test cases t_1 and t_2 are in the same test suite such that the set of faults that t_2 can potentially detect is a subset of the set of faults that t_1 can potentially detect, then t_1 is said to be an effective test case, and t_2 is said to be a redundant test case. A randomly generated test suite is likely to contain a high ratio of redundant test cases. Thus random test case generation is not guaranteed to produce an effective test suite. For this reason, various test generation strategies are employed that enhance the effectiveness of the generated test suite. Traditional black box testing strategies are,

- Equivalence class partitioning
- Boundary value analysis
- Category-partition method

- Partition analysis
- Cause-effect graphing, and
- Error guessing

In equivalence class partitioning [Mye79], each input domain is divided into sub-domains such that the expected behavior of the software is uniform in a sub-domain. For each input variable, a typical value is selected from a sub-domain, and all such combinations of values of the variables are tested. Boundary value analysis suggests that the input variables should be tested at their boundary values. The underlying assumption in boundary value analysis is that the boundaries are implemented as decision predicates in the software, and because of predicate errors the chances of errors at the boundary values of the input variables are greater.

The category-partition method was first introduced in 1988 by Ostrand and Balcer [OB88]. In this method, an input domain is divided into categories on the basis of characteristics of the inputs. Each category is then partitioned into a set of disjoint choices. Test cases are generated so as to cover each combination of choices for the input variables. Partition analysis is a widely used strategy in formal specification based software testing. It is based on partitioning of the pre-condition predicate of an operation, such that a solution to each partition represents a solution to the entire predicate. The most widely used method to partition a predicate is to convert the predicate expression into DNF (Disjunctive Normal Form) or CDNF (Canonical Disjunctive Normal Form).

In cause-effect graphing, a graph is constructed from the requirements specification by identifying the causes (input conditions) and effects (output conditions) [Elm73]. The graph describes the relationships between the causes and the effects using the Boolean logic operators. Test cases are then generated from the graph. Error guessing [Mye79] is an informal strategy in which the tester uses his/her intuition and experience to guess certain types of errors and lists them. Test cases are constructed from the list of probable errors so as to expose the errors.

Formal specification based testing approaches have been classified into the following three categories by Offutt et al. [OL99],

- Model based approaches
- State based approaches, and
- Property based approaches

Model based approaches are based on formal specification notations such as Z, ObjectZ, VDM-SL and VDM++ which are based on the set theory and support modeling of the software in terms of set-theoretic models. State based approaches are based on formal notations which model the system as a state transition machine such as the SCR formal language. Property based approaches are based on the formal specifications which do not model the system but instead define the relationships between functions through the use of axioms. Algebraic specifications are a typical example of system formal specifications.

2.2 VDM++ Specification Language

VDM-SL [Daw91] [Jon90], one of the few formal languages whose syntax and semantics have been completely formally defined, is a model-based specification language based on denotational semantics. VDM++ [DK92] is an object-oriented extension of the ISO VDM-SL. It supports various forms of abstraction, and step wise refinement of abstract models into a concrete implementation. In VDM++, representational abstraction is supported by mathematical data structures, such as sets, sequences, maps, composite objects, Cartesian products and unions. At a lower level, the language provides various numeric types, the Boolean type, tokens and enumeration types. By using the data-structuring mechanism and the basic data types, compound data types can be formed with a specific mathematical structure. Subtyping is supported by attaching domain invariants to domain definitions.

Operational abstraction is supported in VDM++ by function specification, and the operation specification. Both functions and operations may be specified implicitly using pre and post conditions, or explicitly using applicative constructs to specify functions and imperative constructs to specify operations. Operations have direct access to a collection of global objects – the state of the specification. The state is constructed as a composite object, built from labeled components.

A VDM++ specification typically consists of a collection of classes. Each class has a state description, domain definitions, constant definitions, a collection of operations and a collection of functions. An initial specification should be as abstract as possible. Two

techniques are available for further development of the initial specification: data reification, which addresses the refinement of the state elements, and operation modeling, which addresses the refinement of the functions and operations. Data reification involves the transition from abstract to concrete data types, and a justification of this transition. Choosing a more concrete data model implies a redefinition of all operations and functions on the original model in terms of the new model, a process called operation modeling [PKT92].

In VDM++, the top-level system specification consists of a collection of related classes. The body of each class contains the following optional elements,

- *type definitions*
- *value definitions*
- *instance variable definitions*
- *operation definitions*
- *function definitions*
- *synchronization definitions*
- *thread definitions*

For each class, the header specifies class name and optional super class name(s) using *is subclass of* clause with the class name. For example,

```
class A is subclass of B
    ...
```

```
    ...
end A
```

When a class is defined as a subclass of an already existing class, the subclass definition introduces an extended class, which is composed of the definitions of the superclass, and the definitions of the newly defined subclass. The interface to the objects of the subclass is the same as the interface to its superclass extended with the new definitions within the subclass. A subclass inherits from the superclass all of its value and type definitions, instance variables, operation and function definitions, and synchronization definitions.

A class may inherit from more than one superclasses. However, a *name conflict* occurs when two constructs with the same name and of the same kind are inherited from different superclasses. Name conflicts must be explicitly resolved through *name qualification,* i.e. prefixing the construct with the name of the superclass and a `-sign (back-quote).

Polymorphic behavior cannot be explicitly specified in VDM++. However, a variable *v* of the superclass can be assigned an object of a subclass which allows overriding methods of the subclass to be invoked through *v*. Moreover, in a superclass, it is possible to delegate the responsibility to define an operation to the subclass(es) by using the *is subclass responsibility* clause, e.g.,

```
class A
     …
     operations
     op1() is subclass responsibility
end A
```

The operation *op1()* is defined by a subclass *B* derived from class *A*. A superclass containing one or more abstract operations acts as an *abstract base class*.

In this thesis, we use the CSK VDM++ [CSK05] for specification based test case generation. CSK VDM++ was originally called IFAD VDM++, developed at IFAD, Denmark. A tool called *VDMTools* is also available for CSK VDM++ that supports syntax checking and consistency analysis of a VDM++ specification.

2.3 Unified Modeling Language

The Unified Modeling Language (UML) [OMG05a] is widely recognized as a standard graphical notation for specifying the structure as well as behavior of object-oriented systems. It consists of a set of constructs common to most object-oriented languages. However, the semantics of the UML have not been formalized [EFLR98]. While the Object Management Group (OMG) was responsible for the standardizing of the UML as a notation, the semantics of the UML is still a research issue [PB00]. Despite this limitation of the UML, the formal methods community has come to recognize the role of the UML in decomposing complex problems and presenting abstract visual perspectives of the models [AS99] [ELCKAH98] [HCKKTFSMSCM95]. The use of the UML along with a formal specification offers complementary benefits such as visualization of the

models, and providing higher-level structural views of the system, while the formal notations can fill in the processing details with their precise and unambiguous syntax [AS99].

In an object-oriented system, related classes interact with each other to provide some system functionality. Even though individual classes may function correctly, integration of classes can result in several new kinds of errors such as interface errors, conflicting functions, missing functions [Bin99]. Thus, testing of all possible interactions between collaborating classes is required to ensure correct functionality of the system. However, an implicit specification in VDM++ does not model the interactions between classes.

In recent years, the researchers have realized the potential of UML in software testing. Many of the UML design artifacts have been used in different ways to perform different kinds of testing. For instance, UML state-charts have been used to perform unit testing, and interaction diagrams (communication and sequence diagrams) have been used to test class interactions. In this dissertation, we propose a novel technique that combines a UML communication diagram with the formal specification to automatically generate test paths for integration testing of object-oriented systems. We use the UML communication diagrams to generate message sequences, and the formal specification to generate state invariant predicates for the states in which each class can receive a message. The message sequences are combined with the state invariants to construct test models, which are used to generate the test paths.

Chapter 3

Formal Specification Based Testing Techniques

A survey of the literature reveals a large volume of research work in the area of formal specification based software testing. However, the existing testing techniques focus only on unit testing – the use of formal specifications in object-oriented integration testing is still a relatively unexplored area. In this chapter, a survey of the testing techniques based on formal specifications is presented. The survey comprises of a brief description of each testing technique followed by its analysis.

3.1 Evaluation Criteria

We define the following criteria as a basis for analysis of the existing formal specification based testing techniques:

- *Object-orientation:* OO paradigm is the de facto industry standard for software development. In order to support object-oriented development, the existing model-based formal specification notations have also been extended with OO constructs. In our analysis, we consider whether the testing technique supports

object-oriented paradigm or not, i.e., whether the formal specification language used for generating test cases allows specifying an object-oriented system. If a structured (non object-oriented) formal notation is used, it is considered whether the test generation approach can be adapted to object-oriented specification or not.

- *Testing level:* This considers the level at which a testing technique is applicable, i.e., whether the test cases are generated for class testing, integration testing, or system level testing.
- *Strategy Flexibility:* This parameter considers that if the testing technique is restricted a particular test generation strategy or it allows the tester to select a strategy. Strategy flexibility is an important parameter for analysis of testing techniques, since it allows the tester to apply different strategies to generate test cases, thus allowing more thorough testing. Some commonly used black-box testing strategies have been described in Chapter 2.
- *Notation Adaptability:* Most authors demonstrate their testing techniques for a particular formal specification language. However, a more adaptable testing technique should be applicable to multiple formal notations without major changes in the technique itself. The notation adaptibility parameter considers whether a testing technique can be easily adapted to some other formal specification language or not.
- *Automation:* This parameter considers whether the proposed testing technique can be automated, and if so, whether full automation is possible or only partial automation can be done; do the authors provide algorithms for automation? If a

technique is fully or partially automatable, then it is considered whether or not it is supported by a tool, and whether the tool is a prototype or a complete version.

3.2 The Testing Techniques Surveyed

This section surveys the formal specification based testing techniques and gives a brief analysis against each technique using the evaluation criteria defined above. The techniques appear in chronological order by the year of publication.

3.2.1 Hall, 1988

Hall proposed a test case generation technique [Hal88] based on the Z specification language. The proposed technique used partition analysis strategy to divide the input space into partitions, for test generation purposes. A brief analysis of Hall's technique is as below:

- Z is a structured specification notation, so the proposed technique does not cater to object-oriented paradigm.
- The testing technique proposed by Hall is based on partitioning of input predicates, which is a generic idea, i.e., it can be easily applied to other formal notations based on first-order logic.
- The partition analysis strategy used in the proposed approach can be generalized to other test generation strategies as well.
- The testing technique is described at an abstract level, and there is no discussion on whether it can be automated or not. Also, there is no tool support for the proposed technique.

3.2.2 Tsai, Volovik & Keefe, 1990

Tsai, Volovik and Keefe proposed a testing technique [TVK90] based on relational algebra specifications. Their proposed technique transforms relational algebra expressions to predicates which are further converted to systems of linear equations. Test cases are then generated by solving these systems of linear equations. The authors have defined mapping rules for transformation of relational algebra expressions to predicates and predicates to systems of linear equations. They also propose automation of their technique and demonstrate the results. The following is a brief analysis of the Tsai et al.'s testing technique:

- Since the proposed technique is based on relational algebraic specifications, it cannot be easily generalized to the model-based formal specification languages.
- There is no support for object-orientation, and it is not obvious whether the technique can be extended for testing of object-oriented systems.
- The testing strategy used is the domain testing, however, the proposed technique is flexible and allows other testing strategies to be used as well.
- Although automation of the technique has been proposed, no tool support exists.

3.2.3 Doong & Frankl, 1991

Doong and Frankl proposed a set of tools for object-oriented testing [DF91] called as ASTOOT. The tool set includes tools for test case generation, test driver generation, test case execution and evaluation of the results. It is based on an algebraic specification of abstract data types and the specification language used is LOBAS. Their approach is based on testing interactions among operations of a class, and is demonstrated on two case studies. A brief analysis of the Doong and Frankl's approach is as below:

- The proposed technique supports object-orientation, and has been demonstrated on two case studies.
- Even though it tests interactions between the operations of a class, the technique is applicable at the class level testing only, i.e. it does not test interactions between the objects of a system.
- Doong and Frankl's work is based on an algebraic specification language LOBAS, and cannot be easily extended for other specification languages. However, it is flexible enough to be used with different testing strategies.
- The proposed technique is automatable and is also supported by a tool that generates test cases and the test driver.

3.2.4 Amla & Ammann, 1992

Amla and Ammann apply the category-partition method to the Z specification [AA92]. They apply the category-partition method to obtain a TSL (Test Specification Language) script from a Z specification. Their work shows that TSL can be obtained from a formal specification more easily than from an informal requirements specification. The following is a brief analysis:

- Amla and Ammann's technique is based on structured Z specification and applies to unit level testing only. It can be applied to other model based formal notations as well, however, a major issue to be considered is how a TSL specification can be obtained from other formal specification notations.
- As Z is not an object-oriented specification language, so the issues in OO testing have not been discussed. However, the technique can be extended to class level testing for object-oriented systems.

- The test generation strategy used in Amla and Ammann's work is the category-partition method which ensures proper specification coverage.
- Amla and Ammann have not shown how their technique can be automated and no tool exists to support the technique. However, generation of TSL scripts can be automated.

3.2.5 Dick & Faivre, 1993

Dick and Faivre [DF93] proposed a methodology to convert VDM-SL precondition expressions into disjunctive normal form (DNF), so that a solution to each disjunct represents a solution to the entire expression. A finite state automaton (FSA) is then constructed from the partition predicates, and a Prolog tool is used to derive test cases from the FSA under a given coverage criterion. A brief analysis of the technique is as below:

- The technique proposed by Dick and Faivre has been frequently referenced in several works, and has also been extended in several works by other researchers. Although the technique was originally demonstrated on VDM-SL specification, it can be easily applied to other formal notations that are based on first-order logic.
- It has been demonstrated for unit level testing, i.e., the test cases are generated for testing of individual operations. The issues of object-oriented testing have not been discussed, however, the proposed technique can be extended to class level testing of object-oriented systems.
- Dick and Faivre's technique is partially automatable and is supported by a Prolog tool. However, a major limitation of this technique is that it suffers from the state explosion problem, and is thus not scalable to larger systems.

3.2.6 Laycock, 1993

Laycock [Lay93] presents a case study that uses Ostrand and Balcer's category-partition method [OB88] to generate test cases from a Z specification. The author applies the category-partition method to generate test cases for a single function for which categories can be identified and partitions can be made. However, the technique does not discuss how category-partition method may be applied to a complex system involving interactions between functions and the situations where categories and partitions cannot be easily identified. A brief analysis of the proposed technique is presented below:

- The proposed technique is independent of the specification language, however, it requires the tester to identify categories and partitions.
- It applies to unit level testing only, and the authors do not discuss its application to testing of object-oriented systems. However, the proposed technique can be extended to class level testing for OO systems.
- Since this work is highly dependent on the category-partition method, it cannot be easily adapted for other testing strategies.
- The technique can be partially automated but is not supported by any tool.

3.2.7 Weyuker, Goradia & Singh, 1994

Weyuker, Goradia and Singh proposed a technique [WGS94] to generate test data from a Boolean formula by converting it into canonical disjunctive normal form (CDNF). The proposed technique is not notation-specific and can be applied to any formal notation that is based on the first-order logic. Their technique generate both positive and negative test

cases with sufficient coverage of the specification. A brief analysis of Weyuker et al.'s technique is presented below:

- The proposed technique is independent of the specification language, however, it requires the specification to be represented as Boolean formulas.
- It applies to unit level testing only, and the authors do not discuss its application to testing of object-oriented systems. However, the proposed technique can be extended to class level testing for OO systems.
- The testing strategy used in Weyuker et al.'s work is partition analysis. Since their work is highly dependent on this strategy, it cannot be easily adapted for other testing strategies.
- The technique can be automated and is also supported by a tool.

3.2.8 Blackburn & Busser, 1996

Blackburn and Busser [BB96] proposed a technique to generate test cases from state-based formal specifications. Their technique is based on deriving the constraints from functional relationships between inputs and outputs, and then solving the constraints to generate test cases. Their technique is supported by a tool called T-VEC (Test VECtor) that automatically test cases consisting of the input values and their expected outputs. Following is a brief analysis of Blackburn and Busser's technique:

- The proposed technique has been demonstrated for state-based formal specifications, and it is not obvious if it can be applied to the model-based formal specification languages or not.
- The issue of object-orientation has not been discussed, and apparently the technique cannot be easily adapted for testing of object-oriented systems.

- The technique is based on the partition analysis strategy, but again it does not have the flexibility to be adapted for other testing strategies.
- A major advantage is that the technique is fully automatable and is also supported by a tool called T-VEC.

3.2.9 Stocks & Carrington, 1996

Stocks and Carrington proposed a test template framework which uses the Z notation to generate test templates [Sto93] [SC93] [CS94] [SC96]. This work was further extended by Carrington et. al. [CMMMS00] for specification-based class testing. Carrington et. al. show their proposed framework to be flexible by allowing test generation strategy to be specified. However, their framework has not been fully implemented. Following is a brief analysis of their work:

- The authors have used the Z specification language to demonstrate their approach, however, the technique presented is not specific to any particular notation and can be easily applied to other formal notations.
- The original technique proposed in [CS94] and [SC96] was based on structured Z specification, but later it was extended for object-oriented notations and demonstrated using Object-Z. However, it is limited to the class level testing only.
- The proposed framework does not restrict itself to any particular testing strategy, and is flexible enough to adapt to any specified strategy.
- The test template framework proposed by Stocks and Carrington can be partially automated, however, no tool support exists.

3.2.10 Helke, Neustupny & Santen, 1997

Helke, Neustupny and Santen [HNS97] describe the use of a theorem prover tool Isabelle to automate the generation of test cases from Z specifications encoded in Isabelle/HOL. The tool converts Z predicates to DNF, eliminates unsatisfiable disjuncts, and generates valid test cases by searching the state space.

- Helke et al.'s technique applies to the Z specification language which does not cover object-orientation. It is not obvious how the technique can be extended for object-oriented formal specifications.
- It applies to the unit level testing only.
- The testing strategy used is partition analysis, however other strategies may also be used.
- It is automatable and is supported by a tool.

3.2.11 Hierons, 1997

Hieron's work on Z specification based testing [Hie97] demonstrates a formal analysis of the Z specification to produce testing related information. He also gives an algorithm that rewrites a Z specification into a form that can be used to partition the input domains for test case generation, and to derive the states of a finite state automaton (FSA) used to control the testing process. The algorithm given by Hierons rewrites the Z specification as a first-order predicate calculus expression in the form of a conjunction of the preconditions with postconditions, for each operation. These predicate calculus expressions are then separated into input predicates and output predicates, and then partitioned into sub-domains. Test cases are then generated for each sub-expression and from each sub-domain. Hierons recognizes that the problem of scale may arise if the

technique is applied to a large system, and discusses the abstraction and independence techniques to alleviate this problem. However, this problem is still open to further research. Hierons also suggests that a tool similar to the one developed by Horcher and Peleska can be developed for his algorithm, to partition the predicate logic expressions.

An analysis of the Hierons' technique is given below:

- Hierons' work is confined to the structured Z specifications only, and no further work has been done to extend it to object-oriented formal notations. However, the ideas of conjoining pre and post conditions for the Z operation schemas, and the construction of an FSA can be applied to class level testing of an object-oriented system. But further research is required to explore the feasibility of applying these ideas to classes.
- It is not clear as to whether the Hierons' technique can be easily applied to other formal notations. Further research is required to explore how his ideas might apply to VDM-SL and other model-based specification languages.
- The test generation strategy used in Hierons' work is the domain propagation strategy. The technique lacks flexibility, since it does not support the other specification-based test generation strategies.
- Another problem as mentioned earlier, is that of the scale, i.e., the proposed technique may not be feasible for large systems due to the problem of state explosion. Moreover, the performance of the algorithm may become degraded for larger systems.

- The proposed technique supports test sequence generation and generation of partitions, but does not support generation of test data.
- The technique is not supported by a tool, however, it can be automated in part at least.

3.2.12 Richardson & O'Malley, 1997

Richardson and O'Malley's approach [RM97] is based on Anna [LH85] and Larch [GHW85] specification languages. The basic idea proposed by Richardson and O'Malley is to apply the code based testing approaches to formal specifications. They propose two kinds of testing strategies, i.e., error-based and fault-based strategies, and apply them to Anna and Larch specifications. The following is a brief analysis of Richardson and O'Malley's approach:

- Richardson and O'Malley's technique applies to structured formal specifications only, however, it can be adapted for object-oriented specifications. The level at which the technique is applicable is the unit level.
- Although Richardson and O'Malley have demonstrated their testing approach for Anna and Larch specification languages, it can be easily applied to other formal languages as well.
- Their work extends the white-box testing strategies so that they can be applied to formal specifications. A major advantage of their technique is its flexibility to adapt to a number of different black-box and white-box testing strategies.
- No tool support is provided, however, the technique can be partially automated.

3.2.13 Singh, Conrad & Sadeghipour, 1997

Singh et al.'s work [SCS97] is based on the Z specification language. They use the classification tree method to organize the input predicates, and then use the disjunctive normal form (DNF) to construct the disjuncts. The classification tree method is based on the well-known category-partition method [OB88]. It is a partitioning method that partitions the input domains with respect to the classifications that are relevant to testing. The input domain is kept at the root of the tree, and the lowest-level classifications appear at the leaf nodes of the tree. These classifications become headers of a combination table that represents the partitioning information, such that each row of the table corresponds to a test case. The following is an analysis of the testing technique proposed by Singh et al.

- The technique proposed by Singh et al. applies to the Z specification language, and combines the classification tree method and the DNF, both of which can be generalized to other formal notations.
- It gives a thorough coverage of the specification by extracting the predicates from the Z specification, and partitioning them.
- Since the technique is based on Z specification language, as such it does not support object-oriented testing and is limited to unit level testing only.
- Singh et al.'s testing technique can be partially automated, and is also supported by a tool called CTE.

3.2.14 Meudec, 1998

Meudec [Meu98] proposed a method to generate test cases from VDM-SL specifications by converting the pre and post condition expressions into DNF, partitioning the DNF into equivalence classes and using boundary value analysis to generate test cases from the

equivalence classes. The approach is based on parsing VDM-SL expressions, and was later implemented by Atterer [Att00]. A brief analysis of this approach appears below:

- The proposed is applied to VDM-SL specification, but it can be extended to other formal specification notations as well.
- However, it does not address the issue of object-orientation, and it is not clear whether the technique can be easily extended for testing of object-oriented systems.
- It applies at the unit testing level only, and there is no discussion on how it can be extended for object-oriented testing.
- It employs partition analysis and boundary value analysis strategies, but other black-box strategies may also be used.
- The technique is shown to be automatable and a tool was implemented by Atterer [Att00].

3.2.15 Offutt & Liu, 1999

Offutt and Liu propose a test generation technique [OL99] for the SOFL specification language. The SOFL is a formal development methodology that combines structured and object-oriented methodologies with formal methods. A SOFL specification consists of three types of components called as a Condition Data Flow Diagram (CDFD), S-modules, and I-modules. The technique proposed by Offutt and Liu uses all three components to generate the test cases. A brief analysis of the technique follows:

- The proposed technique does support some aspects of object-oriented paradigm since SOFL is based on combination structured and object-oriented

methodologies. However, it is highly dependent on the structure of a SOFL specification, and cannot be easily generalized for other formal notations.

- It supports both unit level as well as integration level testing.
- The proposed technique uses partition analysis strategy on the predicates, however, other testing strategies may also be used.
- Although no tool is available for the proposed technique, it can be almost completely automated.

3.2.16 Boyapati, Khurshid & Marinov, 2002

Boyapati, Khurshid and Marinov presented a framework [BKM02], called Korat, that uses Java Modeling Language (JML) predicates to generate the input space, and a *finitization class* to bound the input state space. The bounded state space is searched and invalid objects are discarded. The authors have implemented their proposed framework, and have shown it to be efficient and effective, but its main limitation is that it is Java specific. The following is a brief analysis of their work:

- Boyapati et al.'s work is based on the Java Modeling Language, and as such it does support object-orientation, but the test cases are generated for class level testing only.
- A major limitation of this work is that it is based on the JML specification which is embedded within a Java program, thus it cannot be generalized and adapted for other formal notations.
- The proposed technique is fully automatable and is also supported by a tool named Korat.

3.2.17 Liu, Miao & Zhan, 2002

Liu, Miao and Zhan [LMZ02] further extended the work of Stocks and Carrington [SC96] for object-oriented specifications. It is based on Object-Z notation, and can be partially automated. The proposed framework in [LMZ02] generates a valid input space (VIS) for class methods, and applies a strategy on VIS to generate test data. Valid sequences of execution of methods are determined by constructing a finite state machine (FSM) for the class under test. The following is a brief analysis of their work:

- As an extension of Stocks and Carrington's work for Object-Z notation, it does support the object-oriented paradigm. However, the test cases are generated for class level testing only.
- As in Stocks and Carrington's work, the proposed testing technique is independent of the testing strategies, i.e. any specified strategy can be used to generate test cases.
- The proposed framework is partially automatable, and is also partially implemented.

3.2.18 Bernard, Legeard, Luck & Peureux, 2004

Legeard and Peureux present a case study on generating test sequences for Smart Card GSM 11-11 standard [BLLP04] to evaluate the effectiveness of B Testing Tools (BTT) testing environment on a large real-life application. In another paper, Legeard, Peureux and Utting [LPU02a] compare the BTT testing environment with the TTF framework of Stocks and Carrington [SC96]. The test generation method used in the BTT environment is based on the B notation. The testing approach is based on computation of all the boundary states for the B machine (a boundary state is defined as a state in which at least

one state variable has the minimum or maximum value), and generating a test path for each boundary state. The test paths (called preambles) ensure that a boundary state is reached from the initial state. The operation to be tested is then invoked from each boundary state and the final state is examined. The authors demonstrate that the BTT method gives a wide coverage (compared with manually generated tests) and saves 30% of test design time.

In BTT method, the preamble is computed automatically using a best-first search algorithm on a constrained reachability graph. The authors mention a limitation of this approach as being based on the assumption of uniformity on the domain of the path. Another limitation is that only the first path discovered by the algorithm is used as the preamble. As there can be multiple paths (possibly infinite) leading to a boundary state from the initial state, the single path coverage may not be adequate. A brief analysis of the approach follows:

- The BTT generates test cases for an operation of an abstract machine specified using the B notation. Although it does not directly support object-orientation, the technique can be easily extended to class level testing of object-oriented systems.
- It is based on the B method but can be applied to other model-based formal notations as well.
- The testing strategies used are partition analysis and boundary value analysis, however, other similar strategies may also be used.
- The proposed technique is highly automatable and is also supported by a tool.

3.2.18 Miao & Liu, 2006

Miao and Liu [ML06] extended the work of Stocks and Carrington [SC96] and Liu et al. [LMZ02]. They proposed a test class framework for object-oriented class testing using Object-Z specification. Their proposed framework is partially automatable. It generates a valid input space (VIS) for class methods, and applies a testing strategy on VIS to generate test data. Valid sequences of execution of methods are determined by constructing a finite state machine (FSM) for the class under test. The following is a brief analysis of their work:

- It supports the object-oriented testing. However, the test cases are generated for class level testing only.
- As in Stocks and Carrington's work, the technique proposed by Miao and Liu is independent of the testing strategies, i.e. any specified strategy can be used to generate test cases.
- The test class framework is partially automatable, and is also partially implemented.

3.3 Conclusion

From the analysis of existing formal specification based testing techniques given in section 3.2, we make the following conclusions:

- the existing formal specification based testing techniques primarily focus on unit level testing only, while integration testing is largely ignored.
- most of the existing techniques have been developed for structured systems – only a few of them support testing of object-oriented systems.

- the techniques that support testing of object-oriented systems focus on class level testing only. Formal specification based integration testing of object-oriented systems is still a research issue.
- while inheritance and polymorphism are powerful features in object-oriented paradigm, the existing formal specification based testing techniques almost completely ignore testing of inheritance and polymorphic relationships in object-oriented systems.

The above conclusions motivated us to explore the possibility of integration testing using formal specifications, and to develop a comprehensive framework for object-oriented testing based on formal specification. We chose VDM++ as the specification language since it has well-defined syntax and semantics, and it is also supported by a tool called *VDMTools*.

Chapter 4

UML Based Integration Testing Techniques

In object-oriented paradigm, a class is considered as a unit, which is the focus of unit testing. Integration testing is concerned with testing of the interactions between classes. The purpose of integration testing is to validate that classes, provide the intended functionality when they are made to interact with each other. While individual classes may be implemented correctly, and thoroughly tested, faults may occur due to their faulty interaction. The interactions between classes are realized by method calls from an object of one class to an object of another class. The implementation of any useful functionality of the system involves interaction between multiple classes. While unit testing considers the state of a single object of a class, integration testing involves considering the states of multiple classes involved at the same time in an interaction. The rapidly growing trend in software development community towards component based development also necessitates efficient and effective integration testing, as the most important part of the development of such systems is integration testing.

UML-based testing techniques have focused on both unit testing and integration testing. UML interaction diagrams, i.e., sequence and communication diagrams (formerly called collaboration diagrams), are used to model the interactions between classes. Even though both types of interaction diagrams are essentially equivalent, the sequence diagrams are commonly used for test generation purpose, communication diagrams have been rarely used for this purpose. In our framework, we use UML communication diagrams together with a VDM++ specification to derive integration level test cases. The information extracted from a collaboration diagram is how the objects interact with each other to provide some functionality, while the VDM++ specification is used to determine the various states in which an object can be while receiving a message. This chapter surveys the UML based integration testing techniques proposed in the literature.

4.1 Abdurazik & Offutt, 2000

Aburazik and Offutt [AO00] used UML collaboration diagrams for test generation, however, their technique is not supported by any tool and no algorithms have been suggested to automate the generation of test cases. Their main contribution was to adapt the conventional data and control flow testing strategies to UML collaboration diagrams. The interactions between the objects in a collaboration diagram are represented by messages, which represent both data and control flow of the operation. Abdurazik and Offutt's strategy uses the flow information for static checking of the source code.

To identify data flow paths in a collaboration diagram, they categorize the links between the objects in a collaboration diagram into six types, i.e. variable definition link, variable usage link, object definition link, object usage link, object creation link, and object

destruction link. Based on these types of links, they identify four types of pairs of links for testing, i.e. variable def-use link pair, object def-use link pair, object creation-use link pair, and object usage-destruction link pair. To identify control flow paths in a collaboration diagram, all possible message sequences, starting from the external message up to the last message, are considered.

A major limitation of this technique is that it only performs static checking while the issue of generating tests from the UML model is not addressed.

4.2 Basanieri & Bertolino, 2000

Basanieri and Bertolino [BB00] proposed an integration testing technique called Use Interaction Testing (UIT) based on use cases, sequence diagrams and class diagram. Their technique performs bottom-up integration testing by conducting a dependency analysis of the use cases to identify the least dependent use cases. For such use cases, test cases are generated by combining the information present in the use cases with corresponding sequence diagrams. Similarly, in the next iteration, the least dependent use cases for the next level are identified and the process of test generation is repeated.

A limitation of this approach is that some activities such as identification of dependencies between the use cases require human judgment and thus cannot be automated.

4.3 Basanieri, Bertolino and Marchetti, 2001

The Basanieri et al.’s work is based on the Basanieri and Bertolino’s work [BB00] described above, i.e. Use Interaction Testing (UIT). The authors propose a technique

called Cost Weighted Testing (COWTest) to prioritize and select test cases from the test suite generated by UIT method. Tests are prioritized and selected on the basis of two criteria, i.e. a fixed level of coverage, or a fixed allowed cost. The technique is supported by a tool called COW Suite that implements the UIT testing technique described earlier and COWTest method. The tool can be used to generate an optimized set of test cases under a specified criterion. However, the tool does not fully automate the test generation process, expert judgment and human intervention are still required in some sub-tasks.

4.4 Pilskalns, Andrews, France & Ghosh, 2003

The technique developed by Pilskalns et al. [PAFG03] is based on the category partition approach [OB88]. It merges behavioral and structural UML models to generate a new test model called Object Method Execution Table (OMET), which captures test sequences and corresponding data values. It is produced by combining UML class and sequence diagrams. Sequence diagrams are used to produce another data structure called Object Method Directed Acyclic Graph (OMDAG) which captures control flow paths for all message sequences through the sequence diagram. Then, class diagram is used to generate the partitions for the attributes and parameters values of the methods of classes involved in the message sequences. The authors also give an algorithm to generate OMETs for each path of the OMDAG. Paths are generated from the OMDAG by graph traversal algorithms. This testing technique can be automated, however, it requires the attribute and parameter domains to be specified in OCL in the class diagram. At the present, however, there is no tool support for this technique.

4.5 Fraikin & Leonhardt, 2002

Fraikin and Lenhardt's testing technique [FL02] is based on the sequence diagrams and is supported by a tool called SeDiTec. The approach is based on model execution that executes a sequence diagram and collects some testing information which is further used to generate an output sequence diagram. The input and output sequence diagrams are then compared for consistency. The sequence diagrams are represented as XMI files in the tool. The tool also supports generation of stubs for methods that have not yet been implemented. This allows the tester to start testing at an early stage. The tool also supports inheritance relationships among the classes. However, the tool is language dependent and only supports Java implementations.

4.6 Wittevrongel & Maurer, 2001

Wittevrongel and Maurer [WM01] propose a testing technique based on the sequence diagrams for scenario-based test case generation. Their basic idea is based on identifying the frequently exercised scenarios from the sequence diagram and generating test cases for each scenario. The actual test inputs and expected outputs are provided by the user. The technique is also supported by a tool which reads the sequence diagrams in XMI format. The tool also generates the test driver and supports automated execution of test cases, however, the tool does not automatically generate test input values and expected outputs.

4.7 Wu, Chen & Offut, 2003

Wu, Chen & Offut [WCO03] propose a test generation approach for component-based systems that is based on UML models. The proposed technique exercises component interfaces with all possible scenarios. Each interface is further exercised by all possible events that invoke the interface. This allows for more thorough testing of the components without a corresponding increase in complexity. The technique allows for testing of context-dependent relationships by exercising the scenarios of UML interaction diagrams. The technique is effective for testing of Commercial-Off-The-Shelf (COTS) components for which source code is not available but UML interaction models can be constructed. However, there is no tool to support this technique.

4.8 Pelliccione, Muccini, Bucchiarone, & Facchini, 2004

Pelliccione et al. present a technique called TEst Sequence generaTOR (TESTOR) that uses UML state-charts and collaboration diagrams to generate test sequences [PMBF04]. TESTOR generates a set of sequence diagrams from these two models, and finally generates test sequences as scenarios from the sequence diagrams. The algorithm used in TESTOR to select test sequences selects only a finite number of output sequences and avoids the state explosion problem.

4.9 Gallagher, Offutt & Cincotta, 2006

In a recent work, Gallagher et. al. [GOC06] extend the idea of class state machines (CSMs) to generate integration tests for multiple classes. The idea of a CSM for class testing was originally proposed by Hong et al. [HKC95], which was based on

constructing a state machine for a single class that modeled the behavior of the class. The CSM was transformed into a data flow graph that explicitly identified the definitions and uses of each state variable of the class, and then applied conventional data flow testing to produce test case specifications that could be used to test the class. The integration testing technique proposed by Gallagher et al. extends this idea to interaction testing of classes. In this technique, CSMs are combined to form data sets according to a defined relational database schema. Database queries are used to extract def-use relationships among the classes, and a component flow graph is then constructed from the transitions related to the components identified for testing. Test paths are generated from the component flow graph by applying various testing criteria. Finally, the feasible test paths are identified, and converted into executable test cases. The proposed technique is automated and the empirical results have also been shown.

4.10 Ali, Briand, Rehman, Asghar, Zafar & Nadeem, 2006

Ali et. al. [ABRAZN06] propose a test generation technique based on UML state-charts and collaboration diagrams. The state-chart for each modal class is flattened and combined with the collaboration diagram to create a test model called State COllaboration TEst Model (SCOTEM). Test paths are generated by traversing the SCOTEM model. The authors also define various coverage criteria on the test model to allow generation of an effective set of test cases without increasing the cost too much. The technique is also supported by a tool called SCOOTER which reads the UML diagrams in XMI format and generates test paths under the specified coverage criterion. A major limitation of the tool is that it requires the test data to be manually generated.

4.11 Conclusion

In summary, there are several integration testing techniques that make use UML interaction diagrams to derive dynamic interactions among the objects. However, to our knowledge, there is no existing work that combines a formal specification language and UML models to comprehensively test class interactions in all possible states. Our integration testing approach (chapter 6) is based on combining the VDM++ formal specification with UML communication diagrams. It is based on the idea presented in Ali et al.'s appraoch [ABRAZN06] described above. However, since we use the VDM++ formal specification to construct the state invariants instead of the UML state-charts, our approach is more flexible and allows construction of state invariants under various strategies. It combines both the UML artifacts and the formal specification to generate test paths for comprehensive integration testing of classes in all possible combinations of their states.

Chapter 5

The *SpecTGS* Framework

This chapter together with the next chapter, presents the proposed test generation framework *SpecTGS*. The framework consists of two main parts (Figure 5.1), for unit and integration level test generation. The framework uses an implicit VDM++ specification and a corresponding implementation in C++ to generate test cases for unit level testing. The C++ implementation is required only to generate concrete (executable) test cases. For integration testing, the framework generates test paths from the UML communication diagrams and the VDM++ specification of the classes involved in collaborations. This chapter gives an overview of the framework, and the unit testing component of the framework; at the end of this chapter, we discuss how the generated test cases may also be used for inheritance and polymorphic testing; the next chapter covers integration testing component of the SpecTGS framework.

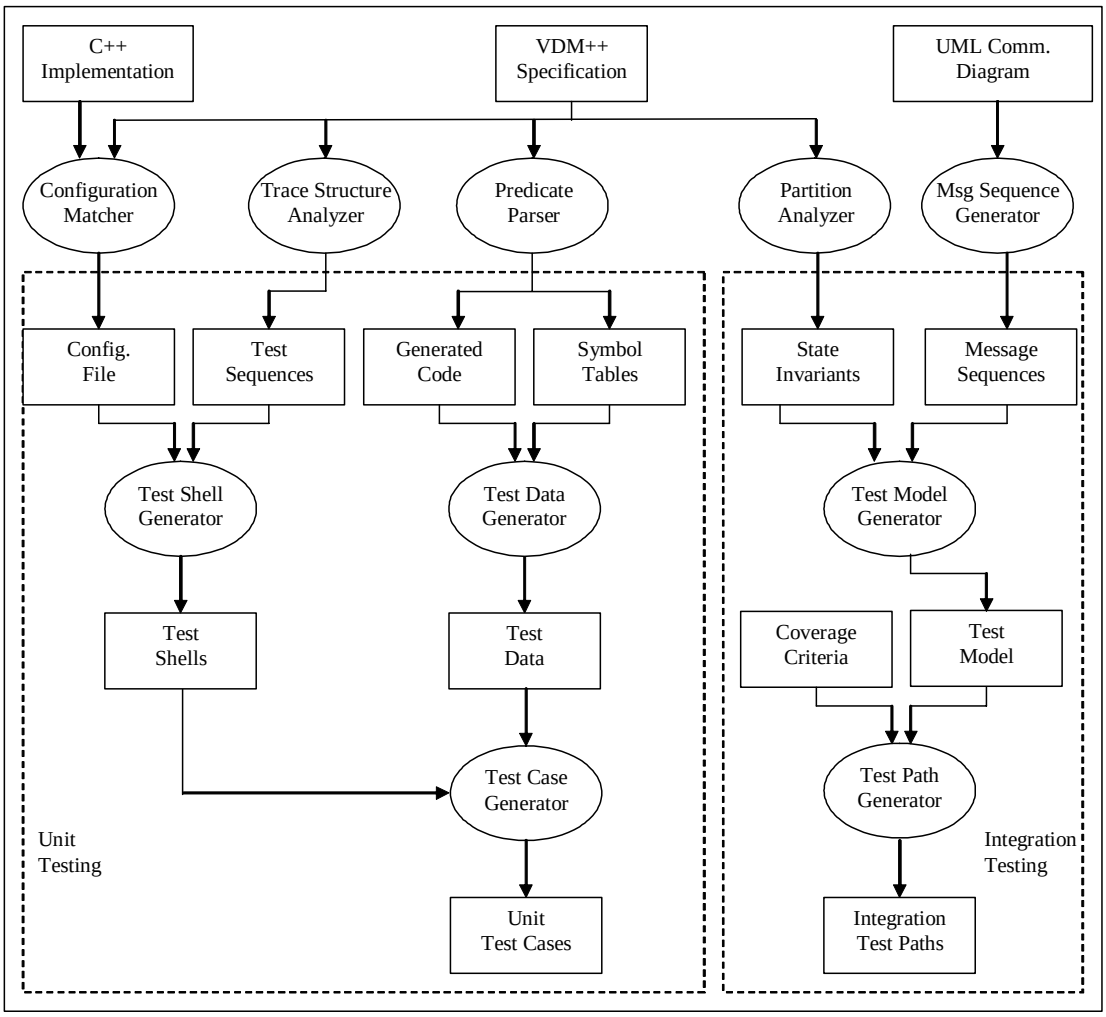

Figure 5.1. Architecture of the SpecTGS

5.1 Class Testing

In object-oriented paradigm, the class is considered as a basic program unit. The framework supports unit testing by generating test cases for a class. The unit testing component of the *SpecTGS* framework consists of the following sub-components,

a) configuration matcher

b) trace structure analyzer

c) predicate parser

d) test shell generator

e) test data generator, and

f) the test case generator

The framework requires a VDM++ specification, and a corresponding C++ implementation. The implementation is required to derive specification-to-code mappings to construct executable test cases. To begin with, the user may choose to generate test cases for an individual method of the class, or for all allowable method sequences of the class. In VDM++, the allowable method sequences for a class can be defined using synchronization constraints. The synchronization constraints may be specified as trace structures using a notation that is based on regular expressions. For the purpose of test generation, we assume that the synchronization constraints for a class are specified as *trace structures* in the VDM++ specification of the class under test. The trace structure analyzer uses these trace structures to generate test sequences. Each test sequence is an allowed sequence of method calls of the class on an object. The actual number of test sequences for a class can be infinite, but our algorithm limits the test sequences to a finite set. The following is a brief description of the framework components:

1. *Configuration matcher* is responsible for mapping names of classes, instance variables, methods, etc., used in the specification with those of the implementation. This process is automated, however the user is allowed to modify or manually create the mappings file.
2. *Trace structure analyzer* constructs valid sequences of the operations of a class from the trace structure specified in VDM++ specification of the class. The trace

structure is defined in the synchronization constraints section of a VDM++ class, and defines valid operation sequences in a notation based on regular expressions.

3. *Predicate parser* constructs a *method entry predicate* and a *method exit predicate* for each method in the class. The method entry predicate for a method is formed by the conjunction of the method pre-condition and the class invariant predicates. Similarly, the method exit predicate for a method is the conjunction of the method post-condition and the class invariant predicates. The parser then generates C++ code for both the method entry and the method exit predicates of each method. At the time of invocation of a method, the class invariant and the method precondition must evaluate to true. The correct behavior of the method under test critically depends on correctness of these predicates. Thus, we can use the method entry predicate to generate input data for a method. Similarly, the correct implementation of a method must result in the post condition being true while the class invariant must remain true after execution of the method. Therefore, the generated code for method entry predicate is used to filter the input data for a method, while the generated code for the method exit predicate is used as an oracle to evaluate the results of method execution for a test case. The parser also creates a symbol table for the method entry predicate, which records variable names and their boundary and typical values.

4. *Test shell generator* combines configuration information with the test sequences to generate *test shells*. It generates empty test shells from the test specification, and the configuration file, which are then filled with test data. A test shell is a sequence of operation invocations, where each operation invocation is a method

call with dummy parameters representing types of the parameters. These dummy parameters are replaced with the test data later, when test shells are converted into concrete test cases.

5. *Test data generator* evaluates the method entry predicate for each method with the data in the symbol tables. The test data are generated from the symbol table created by the parser, and filtered by the method entry predicate.
6. *Test case generator* is responsible for filling the test data in empty test shells. For each input parameter of a method, the test data generator produces multiple test values using boundary value analysis. A test set is defined as a set of values of input parameters for a method. The test sets for a method are formed by taking a cross product of test values for the input parameters. The generated test sets are then filtered by executing them on the code for the method entry predicate.

The test driver is generated as a *child wrapper* class, a subclass of the CUT (class under test), and includes the generated code as public methods in this class. The test driver executes the test cases on the implementation by instantiating the child wrapper, and evaluates results by executing code for the method exit predicate. The rest of this chapter describes the working of each component in greater detail with a running example.

To demonstrate how unit tests are generated, we use a VDM++ specification for a class *NNcomplex* (a simple abstraction of the non-negative complex numbers) and its corresponding C++ implementation as a running example. The VDM++ specification for

NNcomplex class and an implementation in C++ are given in Figure 5.2 and Figure 5.3 respectively.

```
class NNcomplex
  instance variables
    re : real;
    im : real;

  inv (re>=0) & (im>=0);

  operations
    init()
    ext wr re: real
        wr im: real
    post (re=0) and (im=0);

    add(num: int) sum: NNComplex
    ext wr re: real
        rd im: real
    pre num >= -re;
    post (re = re~+num) and (sum=self);

    subtract(num: int) diff: NNComplex
    ext wr re: real
        rd im: real
    pre num <= re;
    post re = re~-num and (diff=self);

    multiply(num: int) prod: NNComplex
    ext wr re: real
        wr im: real
    pre num >= 0;
    post re = (re~*num) and (im = im~*num)
          and (prod=self);

    divide(num: int) quotient: NNComplex
    ext wr re: real
        wr im: real
    pre num > 0;
    post re = (re~/num) and (im = im~/num)
          and (quotient=self);

  sync
    general  T = <(init ; (add* ; subtract* ;
                  multiply* ; divide*)*),
       {init, add, subtract, multiply, divide}>;

end NNcomplex
```

Figure 5.2. VDM++ Specification for *NNcomplex* class

The class invariant *(re>=0) & (im>=0)* specifies that both real and imaginary components of the complex number must be non-negative. Four methods called *add*, *subtract*, *multiply*, and *divide* have been defined for the *NNcomplex* class, to perform the basic arithmetic operations on an *NNcomplex* object with an integer value.

The precondition for each method ensures that the result of operation will be a complex object with both real and imaginary components as non-negative. Precondition for the divide operation also prevents division of the complex object by zero.

```
class Complex {
  private:
    float re;
    float im;

  public:
    void init() {
      re = im = 0;
    }

    Complex add(int x) {
      re += x;
      return this;
    }

    Complex subtract(int x) {
      re -= x;
      return this;
    }

    Complex multiply(int x) {
      re *= x;
      im *= x;
      return this;
    }

    Complex divide(int x) {
      re /= x;
      im /= x;
      return this;
    }
}
```

Figure 5.3. Implementation of the NNcomplex class in C++

5.1.1 Configuration Matching

In order to generate valid test cases for a class implementation, not only types of its attributes and method signatures are required but also names of attributes and methods must be known. As the names used in the implementation may be different from those used in the formal specification, the test generator must maintain mappings between the two to allow test generation from the formal specification. The *configuration matcher*

component of the *SpecTGS* is responsible for mapping names used in the specification with those of the implementation. This process is automated, however the user is allowed to modify or manually create the mappings file.

The configuration matcher first matches class names, by comparing number and types of attributes and method signatures. For instance, class A in specification matches with class B in the implementation if both A and B have the same number and types of attributes, as well as the same number of methods with matching signatures.

Class attributes are matched by their types. Likewise, method names are matched by their signatures (i.e., number and types of parameters). The table in Figure 5.4 below shows how configuration matcher matches VDM++ types with those of C++. Currently, the SpecTGS supports only the VDM++ types shown in Figure 5.4. In the matching process, C++ type qualifiers (*long, short, signed, unsigned*) are ignored if type name is specified, otherwise the type name is assumed to be *int* (the default type in C++).

VDM++ Type	Mapped to (C++ Type)
bool	bool
int	int
nat	int
nat1	int
real	float, double
rat	float, double
char	char
quote type	enum type
seq and seq1 types	array type
map type	array of struct
object reference type	object reference type

Figure 5.4. Mappings of VDM++ types to C++ types

The strategy of matching specification with implementation using types of attributes and method signatures works well in most cases. However, it may fail if two or more attributes in a class have the same type and scope, or two or more methods have the same signature and access specifier. For this reason, the *SpecTGS* prompts the user to confirm each mapping before it is saved to the mappings file. Moreover, the mappings file is saved in the text format, and the user can modify its contents later. Figure 5.5 shows mappings file generated by the configuration matcher for the *NNcomplex* class.

```
class NNcomplex -> Complex

  attributes
    re -> re
    im -> im

  methods
    init() -> init()
    add(int) -> add(int)
    subtract(int) -> subtract(int)
    multiply(int) -> multiply(int)
    divide(int) -> divide(int)
```

Figure 5.5. Mappings identified by the configuration matcher

5.1.2 Trace Structure Analysis

As the correct behavior of a class method may depend not only on the current state of the class object, but also on the correct sequence of messages passed to the object [Bin99], it is necessary to specify the allowable sequences of method calls for the class under test. However, if the correct behavior of a class is not dependent on its message sequences – as is the case in non-modal, or quasi-modal classes [Bin99] – then such a specification may be omitted.

In VDM++, the set of all valid sequences of operations is specified in synchronization constraints in a class specification. The synchronization constraints are usually defined as trace structures. A trace structure defines valid sequences of method invocations of a class for a particular object of the class.

Trace structures are specified using a language based on the notation of regular expressions, together with special trace structure operators, i.e.,

(i) *; (semi-colon)* denotes sequential execution. This is used to enforce the order of execution of operations or operation traces.

(ii) * *(asterisk)* denotes zero or more times repetition of an operation or an operation trace.

(iii) + *(plus sign)* denotes one or more times repetition of an operation or an operation trace.

(iv) ** *(double asterisk)* denotes the projection operator, and is used to restrict a trace of operations to a subset of the operation alphabet.

(v) *w_ (w underscore)* denotes the weave operator, and is used to perform synchronized interleaving of two operation traces.

An implementation of a class with a trace structure specification is correct only if it guarantees that only the specified sequence of invocations can occur. The trace synchronization defines one general trace structure and an arbitrary number of subtrace structures. This scheme allows decomposition of the behavior of an object, in which the general trace structure is built from the subtrace structures.

Since our framework supports only positive testing, it only tests the operation sequences that should be allowed by a correct class implementation. Invalid operation sequences that are not derivable from the trace structure expression are not tested. Thus, the SpecTGS framework does not guarantee that the class implementation will not allow incorrect operation sequences to be executed. In Figure 5.6, we give an algorithm to generate a set of valid operation sequences for a given trace structure. Input to the algorithm is a trace structure expression, and the output is the corresponding set of operation sequences. The following is a brief explanation of the algorithm:

- if an empty expression ε is given as input, the output is the set containing an empty operation sequence.
- if the input expression consists of a single operation *op*, the output is the set containing *op* only, i.e., [*op*].
- if the input expression R is of the form R_1+, then the number of operation sequences formed would be infinite. However, the algorithm generates only a finite number of sequences for up to three iterations of R_1, *i.e.*, R_1, $R_1;R_1$, *and* $R_1;R_1;R_1$.
- if the input expression R is of the form R_1*, then the number of operation sequences formed would be infinite. However, the algorithm generates only a finite number of sequences for up to three iterations of R_1, *i.e.*, ε, R_1, $R_1;R_1$, *and* $R_1;R_1;R_1$.

The trace structure in VDM++ specification of the class *NNComplex* in Figure 5.2 is specified as,

```
general T = <(init ; (add* ; subtract* ; multiply* ; divide*)*),
            {init, add, subtract, multiply, divide };
```

This states that every valid sequence of operations must start with the *init* operation, which can be invoked exactly once. After the *init* operation, the other operations (*add*, *subtract*, *multiply*, *divide*) can be invoked any number of times in any order. For instance, some operation sequences derived from this specification are as below,

```
init
init ; add
init ; subtract ; add
init ; add ; subtract ; divide
init ; multiply ; add
etc.
```

```
function genOpSeqs(R : RegExpr): set of OpSeq
{       OSset : set of OpSeq;
        OSset := [ ];
        if (R is ε) then OSset := [ε];
        else if (R is of the form op) then OSset := [ op ];
        else if (R is of the form R1 ** S) then
                OSset := restrict(genOpSeqs(R1), S);
        else if (R is of the form R1 w_ R2) then
                OSset := weave(genOpSeqs(R1), genOpSeqs(R2));
        else if (R is of the form R1 ; R2) then
                OSset := product(genOpSeqs(R1), genOpSeqs(R2));
        else if (R is of the form R1+) then
                OSset := union(genOpSeqs(R1),
                        genOpSeqs(R1 R1), genOpSeqs(R1 R1 R1));
        else if (R is of the form R1*) then
                OSset := union( [ε], genOpSeqs(R1),
                        genOpSeqs(R1 R1), genOpSeqs(R1 R1 R1));
        return OSset;
}
```

Figure 5.6. Algorithm for generating operation sequences

5.1.3 Predicate Parsing

A formal specification in VDM++ contains pre and post conditions for each method of the class under test (CUT). The predicate parser constructs *method entry and exit predicates* for each method by forming a conjunction of method pre and post condition predicates with the class invariant predicate, as shown below:

$$method\_entry\_predicate = method\_precondition \wedge class\_invariant$$

$$method\_exit\_predicate = method\_postcondition \wedge class\_invariant$$

In addition to the method precondition and class invariant, a *method entry predicate* also includes *type constraints*. For instance, if an input parameter of the method, or an instance variable is of type *nat*, then it is implicitly implied that its value cannot be negative. The method predicates are parsed into parse trees using a context free grammar for VDM++ expressions. The *SpecTGS* implements a simple LR parser to parse the predicate expressions.

5.1.4 Generating Code for Method Predicates

From the parse tree, the parser generates C++ code to evaluate each method predicate. The idea of converting a predicate expression into a parse tree and generating C code from the tree, has been described in [NR04]. Mikk also provides a technique to convert Z predicates to C expressions for evaluation of test results [Mik95]. The parser produces boolean-valued C++ functions named *classname_methodname_pre*() and *classname_methodname_post*() for each method in the CUT. Code generated for the *NNComplex* class is shown in Figure 5.7 below. This code for method entry predicate is

used by the test generator to filter the generated input data and discard the unsatisfiable test cases, while the code for method exit predicate is used by the test driver to evaluate the execution results of the test cases.

```
bool Complex_init_pre(float re, float im) {
  bool result = true;
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_init_post (float re, float im) {
  bool result = true;
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_add_pre(float re, float im, int x) {
  bool result = true;
  result = result && (x >= -re);
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_add_post (float re, float re_old, float im, int x) {
  bool result = true;
  result = result && (re == re_old+x);
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_subtract_pre(float re, float im, int x) {
  bool result = true;
  result = result && (x >= re);
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_subtract_post (float re, float re_old, float im, int x) {
  bool result = true;
  result = result && (re == re_old-x);
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_multiply_pre(float re, float im, int x) {
  bool result = true;
  result = result && (x >= 0);
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_multiply_post (float re, float re_old, float im, float im_old, int x) {
  bool result = true;
  result = result && ((re == re_old*x) && (im == im_old*x));
  result = result && ((re >= 0) && (im >= 0));
  return result;
}

bool Complex_divide_pre(float re, float im, int x) {
  bool result = true;
  result = result && (x > 0);
  result = result && ((re >= 0) && (im >= 0));
  return result;
}
```

Figure 5.7. Code Generated by Predicate Parser

A predicate in VDM++ is a well-formed logical expression that is constructed from clauses and logical connectives. A clause may be a relational sub-expression, or a set membership sub-expression. Also, a clause may be a quantified sub-expression involving universal and existential quantifiers. The conversion of simple relational expressions, and

the expressions involving finite sets, sequences, and maps to C++ is automated in *SpecTGS*. For instance, consider the following set membership expression with a universal quantifier,

forall x in set S & (x<y)

If S is a finite set of elements s_1, s_2, s_3, ……, s_n, then the above expression can be evaluated as,

(s_1<y) and (s_2<y) and (s_3<y) and …… and (s_n<y)

Similarly, an expression with an existential quantifier can be evaluated as,

(s_1<y) or (s_2<y) or (s_3<y) or …… or (s_n<y)

The table in Figure 5.8 shows C++ expressions generated by *SpecTGS* for various types of predicates.

VDM++ Predicate	C++ Expression
a=b	a==b
a<b	a<b
a>b	a>b
a<=b	a<=b
a>=b	a>=b
a<>b	a!=b
not a	!a
a and b	a && b
a or b	a || b
a=>b	!a || b
a<=>b	a==b
a in set S	(a==s_1) || (a==s_2) || (a==s_3) ... where s_1, s_2, s_3, ... are elements of S
a not in set S	(a!=s_1) && (a!=s_2) && (a!=s_3) ... where s_1, s_2, s_3, ... are elements of S

Figure 5.8. C++ boolean expressions for VDM++ predicates

5.1.5 Constructing the Symbol Tables

For each method in the CUT, a symbol table is constructed that stores instance variables, method arguments and their boundary values. The boundary values are determined from method entry predicates. The test generator uses symbol tables to generate test inputs for the methods. For example, for the *add* method of the *Complex* class, the *SpecTGS* generates the symbol table shown in Figure 5.9 below.

Var	Type	Rel. Op.	Boundary Value
re	*float*	>=	*0*
im	*float*	>=	*0*
x	*int*	>=	*-re*

Figure 5.9. Symbol Table for *add()* method

As the boundary value of the variable *x* in Figure 5.9 is dependent on the value of the variable *re*, so the test generator must first generate test values for the variable *re*. A variable may have multiple boundaries if it appears in more than one clauses of the predicate expression. For instance, in the predicate expression *(a>10) & (a<20)*, the variable *a* has two boundary values, i.e., *10* and *20*. For such variables, there are multiple rows in the symbol table.

5.1.6 Generating Test Shells

The test shell generator combines the test sequences (generated by the trace structure analyzer) with the configuration information to construct test shells. A *test shell* is a sequence of test templates, where a *test template* consists of a method name followed by its parameter types. The test shell generator uses mappings from the configuration file to determine method names in the CUT, and saves the generated test shells in a file. For

instance, the following three test shells are constructed from the message sequences generated for the *NNComplex* class.

```
BEGIN TEST 1
 init <>
 add <int>
END TEST

BEGIN TEST 2
 init <>
 add <int>
 subtract <int>
END TEST

BEGIN TEST 3
 init <>
 subtract <int>
 multiply <int>
 add <int>
 add <int>
END TEST
```

5.1.7 Generating Test Data

The test data generator determines *method inputs* for each method in the CUT. Method inputs consist of parameters of the method, including the implicit *this* parameter. It uses the symbol table (section 5.1.5) to generate test values for method inputs, and the code for method entry predicate to filter the test values. Using the boundary value analysis strategy, the *SpecTGS* generates the test values for the *add* method as given in Figure 5.10.

For instance, for the variable *re*, the boundary value is *0*, therefore the generated test values are *0, 1,* and *5*. While the values *0* and *1* are at the boundary, the value *5* is randomly generated from the space *re > 1*. Similarly, test values are generated for the

variables *im* and *x*. A total of 27 sets of test values are thus formed (Figure 5.10) for the *add* method. Each of the generated test sets is then executed on the method entry predicate *Complex_add_pre()* to test if it satisfies method entry predicate or not. The unsatisfiable test sets are eliminated. In our example, all 27 test sets are satisfiable. Unsatisfiable test sets may result if there are variables with multiple boundary values. For variables with multiple boundaries, all boundaries are used to generate test data. However, a test set contains only a single value for each variable. The generated test data are used by the test case generator to construct the concrete test cases.

5.1.8 Generating Test Cases

The test case generator is responsible for filling the test data in empty test shells. For each input parameter of a method, the test data generator produces multiple test values using boundary value analysis. A *test set* is defined as a set of values of input parameters for a method. The test sets for a method are formed by taking a cross product of test values for the input parameters. The generated test sets are then filtered by executing them on the code for the method entry predicate.

For each method in a test shell, the generator generates valid test sets. The empty test shells are then filled in with all possible combinations of test sets for its methods, to form the concrete test cases.

```
1:    re = 0,   im = 0,   x = 0
2:    re = 0,   im = 0,   x = 1
3:    re = 0,   im = 0,   x = 9
4:    re = 0,   im = 1,   x = 0
5:    re = 0,   im = 1,   x = 1
6:    re = 0,   im = 1,   x = 9
7:    re = 0,   im = 8,   x = 0
8:    re = 0,   im = 8,   x = 1
9:    re = 0,   im = 8,   x = 9
10:   re = 1,   im = 0,   x = -1
11:   re = 1,   im = 0,   x = -1
12:   re = 1,   im = 0,   x = -1
13:   re = 1,   im = 1,   x = 0
14:   re = 1,   im = 1,   x = 0
15:   re = 1,   im = 1,   x = 0
16:   re = 1,   im = 8,   x = 12
17:   re = 1,   im = 8,   x = 12
18:   re = 1,   im = 8,   x = 12
19:   re = 5,   im = 0,   x = -5
20:   re = 5,   im = 0,   x = -5
21:   re = 5,   im = 0,   x = -5
22:   re = 5,   im = 1,   x = -4
23:   re = 5,   im = 1,   x = -4
24:   re = 5,   im = 1,   x = -4
25:   re = 5,   im = 8,   x = 6
26:   re = 5,   im = 8,   x = 6
27:   re = 5,   im = 8,   x = 6
```

Figure 5.10. Test values generated for *add()* method

5.2 Setting the Object State

As mentioned earlier, the inputs to a method are not only its explicit parameters, but also the implicit *this* parameter, which represents state of the current object. When testing a method, the current object's state may also have to be set by setting values of its instance variables. By the principle of encapsulation, the instance variables of a class are kept private, so we must add getter and setter methods to the class under test to access and modify values of its instance variables.

Setting values of instance variables is required only when testing an individual method – the object must be in a correct state to accept the message. For example, the *add* message can be accepted only when the real and imaginary components of *Complex* object have defined values, and are non-negative. However, when testing a message sequence, the instance variables are not required to be set. For instance, a valid message sequence requires *add* message to be preceded by *init* message, which will ensure correct object state.

The *SpecTGS* framework supports both individual method testing, and a message sequence testing.

Testing an individual method

When testing an individual method, the values of instance variables *re* and *im* are set via setter methods. For instance, to test the *add* method of Complex class, using test values of Figure 5.10, the following test cases are generated:

```
BEGIN TEST add.1
 set_re <0>
 set_im <0>
 add <0>
END TEST


BEGIN TEST add.2
 set_re <0>
 set_im <0>
 add <1>
END TEST
```

```
BEGIN TEST add.3
 set_re <0>
 set_im <0>
 add <9>
END TEST

etc.
```

A complete set of test cases for the *add* method appears in Appendix-I. It may be noticed that the number of test cases increases exponentially if there are methods with multiple parameters, because, in such a case, all possible combinations of values of parameters are used to generate the test cases.

Testing a message sequence

When testing a message sequence, the object state is not required to be explicitly set – the correct state of the object for each method in the sequence is ensured by its preceding messages. For the example *Complex* class, using test values from Figure 5.10, and test shell 1, the following test cases are generated:

```
BEGIN TEST 1.1
 init <>
 add <0>
END TEST

BEGIN TEST 1.2
 init <>
 add <1>
END TEST

BEGIN TEST 1.3
 init <>
 add <9>
END TEST
```

5.3 Test Driver

For the purpose of testing, a test class can be derived from the CUT as suggested in [TR93]. The derived class is called a *child wrapper*. It inherits all the attributes and methods from the CUT. The extra routines required for testing are added to the child wrapper class, rather than patching an existing class of the system. The test driver instantiates the child wrapper, and invokes its methods to be tested.

The *SpecTGS* implements the strategy described above, i.e., it creates a child class of the CUT and adds its testing methods. Under this mechanism, the class that actually gets tested is the child wrapper rather than the CUT. However, the methods under test are actually implemented in the CUT, so they get tested. This strategy relies heavily on the programming language's inheritance mechanism.

The child wrapper class contains the following additional methods, used for testing:

> *load(TestCase tc)* – used to load a test case from the file; *tc* is the test case number.
> *execute()* – used to execute a loaded test case.

The test driver instantiates the child wrapper to create a test object, and then executes each test case with the test object. The *execute()* method of child wrapper invokes each method in a test case in sequence. For instance, for the test case 1.3 of section 3.3.3, the actual method calls made by *execute()* are:

```
init()
Complex_init_post()
add(9)
Complex_add_post()
```

After execution of each method, the method's *exit predicate* is evaluated, and the results are logged in a file. For failed test cases, the execute() method also logs values of variables for which exit predicate failed.

5.4 Inheritance and Polymorphic Testing

Inheritance and polymorphism are powerful features in object-oriented paradigm that support reusability and dynamic binding, but at the same time, the use of these features in an object-oriented program presents new kinds of challenges to the testers – certain new kinds of faults can arise due to inheritance and polymorphism. Inheritance is a mechanism by which a new class, called the derived class, inherits the attributes and operations of a parent class. The derived class may override some of the functionality of the parent class, by redefining some of the inherited operations. Also, the derived class may add its own methods to implement new functionality. Polymorphism is a mechanism by which functionality of the appropriate derived class is invoked based on the dynamic binding of a derived class object to a parent class variable. Despite the importance of inheritance and polymorphic testing, research on formal specification based testing has largely ignored this area.

A technique for testing inheritance relationships based on flattening of the derived VDM++ specification class has been presented in [NL06]. However, in this section we only discuss how test cases generated by the *SpecTGS* framework can also be used for inheritance and polymorphic testing [NML06]. We use the Offutt et al.'s fault model for subtype inheritance and polymorphism [OAWXH01], which defines nine types of faults due to inheritance and polymorphic interactions.

5.4.1 Specifying Inheritance and Polymorphism in VDM++

In VDM++, the top-level system specification consists of a collection of related classes. The body of each class contains the following optional elements,

- type definitions
- value definitions
- instance variable definitions
- operation definitions
- function definitions
- synchronization definitions
- thread definitions

For each class, the header specifies class name and optional super class name(s) using *is subclass of* clause with the class name. For example,

```
class A is subclass of B
      …
      …
end A
```

When a class is defined as a subclass of an already existing class, the subclass definition introduces an extended class, which is composed of the definitions of the superclass, and the definitions of the newly defined subclass. The interface to the objects of the subclass is the same as the interface to its superclass extended with the new definitions within the

subclass. A subclass inherits from the superclass all of its value and type definitions, instance variables, operation and function definitions, and synchronization definitions.

A class may inherit from more than one superclasses. However, a *name conflict* occurs when two constructs with the same name and of the same kind are inherited from different superclasses. Name conflicts must be explicitly resolved through name qualification, i.e. prefixing the construct with the name of the superclass and a `-sign (back-quote).

Polymorphic behavior cannot be explicitly specified in VDM++. However, a variable *v* of the superclass can be assigned an object of a subclass which allows overriding methods of the subclass to be invoked through *v*. Moreover, in a superclass, it is possible to delegate the responsibility to define an operation to the subclass(es) by using the *is subclass responsibility* clause, e.g.,

```
class A
     …
     operations
     op1() is subclass responsibility
end A
```

The operation *op1()* is defined by a subclass *B* derived from class *A*. A superclass containing one or more abstract operations acts as an *abstract base class*.

5.4.2 Offutt et al.'s Fault Model

Offutt et al. present a fault model [OAWXH01] for subtype inheritance and polymorphism in which they identify nine types of faults due to inheritance and polymorphism. However, only the following four of these fault types can be covered by test cases generated from a VDM++ formal specification, since the specification lacks the detailed information present at the implementation level:

i) *State Definition Anomaly (SDA):* This type of fault can occur if:
   a. an inherited method m_1 of the superclass defines a state variable *v*,
   b. a method m_2 of the subclass that overrides m_1, but does not define the inherited state variable *v* consistently with the overridden method m_1, and
   c. an object *o* of the subclass is assigned to a variable *s* of superclass type, and method *s.m2* is invoked

ii) *State Defined Incorrectly (SDI):* This type of fault can occur if:
   a. an overriding method of the subclass incorrectly defines an inherited state variable, i.e. the computation performed by overriding method is not semantically equivalent to the overridden method, and
   b. an object *o* of the subclass is assigned to a variable *s* of superclass type, and method *s.m2* is invoked

iii) *Incomplete (failed) Construction (IC):* This type of fault can occur if:
   a. the constructor does not define (or incorrectly defines) a state variable *v*,
   b. a method *m* of the class uses the undefined state variable *v*, and
   c. an object *o* of the class invokes method *o.m*

iv) *State Visibility Anomaly (SVA):* This type of fault can occur if:

a. a state variable *v* in the superclass has private access specifier, and

b. an overriding method *m* of a sub-subclass cannot define the inherited state variable of super-superclass due to private access specifier.

5.4.3 The Testing Strategy

The strategy used by the *SpecTGS* for unit level test case generation may also be used to generate test cases for inheritance and polymorphic testing, as follows:

i) For *SDA* and *SDI* faults, only those operation sequences of the subclass are derived from the trace structure which include one or more occurrences of an overriding method. The subclass is then instantiated and the object is assigned to a variable of the superclass type. Each operation of the operation sequences is then invoked with the superclass variable. Examining the state variables after each call to an overriding method would expose any *SDA* or *SDI* faults.

ii) The *IC* fault defined in Offutt et al.'s fault model is not really an inheritance or polymorphism fault, and can be easily exposed by operation sequences that involve a call to the constructor followed by a call to a method that uses a state variable.

iii) The *SVA* fault can occur if the class inheritance hierarchy is at least two level deep. This type of fault can be exposed by executed an operation sequence of the sub-subclass that involves at least one operation that is supposed to define an inherited state variable of the super-superclass.

5.4.4 An Example

As an example, consider the inheritance hierarchy in the UML class diagram for a bank account class, in Figure 5.11. The parent class *Account* is an abstraction of a bank account with the basic attributes and operations common to all types of accounts. The derived classes *SavingsAccount* and *CheckingAccount* model two common types of bank accounts. Figure 5.12a and Figure 5.12b present VDM++ specification for the *Account* and *SavingsAccount* classes, respectively.

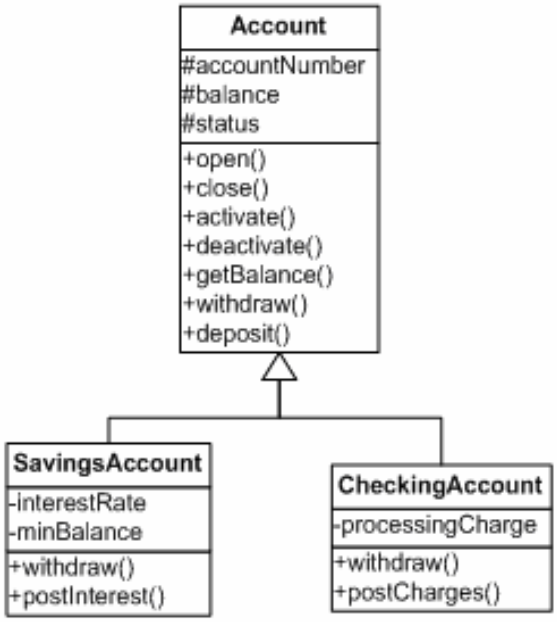

Figure 5.11. Class diagram for Bank Account hierarchy

In an inheritance hierarchy, if synchronization constraints are specified as trace structure expressions in both the subclass and the superclass, the effective trace structure for the subclass is obtained by synchronized weave of the superclass and the subclass trace structures [CSK05].

```
class Account
instance variables
   accountNumber: nat;
   balance: real;
   status: <Active> | <Inactive> | <Closed>
invariant balance >= 100;
operations
  open(amount: real)
     ext wr balance: real;
        wr status: <Active> | <Inactive> | <Closed>;
     pre amount >= 100;
     post balance = amount and status = <Active>;
  close()
     ext wr status: <Active> | <Inactive> | <Closed>;
     post status = <Closed>;
  activate()
     ext wr status: <Active> | <Inactive> | <Closed>;
     post status = <Active>;
  deactivate()
     ext wr status: <Active> | <Inactive> | <Closed>;
     post status = <Inactive>;
  getBalance() bal: real
     ext rd balance: real;
     post bal = balance;
  withdraw(amount: real)
     ext wr balance: real;
     pre balance >= amount;
     post balance = balance~ - amount;
  deposit(amount: real)
     ext wr balance: real;
     pre amount > 0;
     post balance = balance~ + amount;
sync
  subtrace X = <(withdraw* ; deposit* ; getBalance*),
            {withdraw, deposit, getBalance}>;
  subtrace Y = <(deactivate ; getBalance* ; activate),
            {deactivate, getBalance, activate}>;
  general  T = <(open ; (X* ; Y*)* ; (deactivate* ; close)),
           {withdraw, deposit, getBalance,
              deactivate, activate, open, close}>;
end Account
```

Figure 5.12a. VDM++ specification for the Account class

```
class SavingsAccount is subclass of Account
instance variables
   interestRate: real;
   minBalance: real;
invariant balance >= minBalance;
operations
  withdraw(amount: real)
     ext wr balance: real;
     pre balance >= minBalance + amount;
     post balance = balance~ - amount;
  postInterest()
     ext wr balance: real;
     post balance = balance~ * (1+interestRate);
sync
 general  T = <((withdraw* ; postInterest*)*) ,
               {withdraw, postInterest}>;
end SavingsAccount
```

Figure 5.12b. VDM++ specification for the SavingsAccount class

For the above example, the effective trace structure of the *SavingsAccount* class is computed by a synchronized weave of the trace structures of *Account* class and *SavingsAccount* class, as shown below:

```
T =    <((open ; ((withdraw* ; deposit* ; getBalance*)* ;
       (deactivate ; getBalance* ; activate)*)* ;
       (deactivate* ; close)) w_ ((withdraw* ; postInterest*)*)),
       {withdraw, deposit, getBalance, deactivate, activate,
       open, close, postInterest}>;
```

Since *withdraw* method of the *SavingsAccount* class overrides the *withdraw* method of the *Account* class, and also it defines the inherited state variable *balance*, therefore it should be tested for *SDA* and *SDI* faults. To achieve this, the operation sequences containing the *withdraw* operation are derived from the above trace structure and are tested. For example, some of the operation sequences to be tested include,

```
open ; withdraw ; getBalance ; close
open ; withdraw ; deposit ; getBalance ; deactivate ; close
open ; withdraw ; deposit ; postInterest ; close
open ; withdraw* ; getBalance ; postInterest ; close
etc.
```

Chapter 6

Integration Testing with SpecTGS

This chapter covers the integration testing part of the *SpecTGS* framework that combines a UML communication diagram, and the VDM++ formal specification to automatically generate test paths for integration testing. We use the UML communication diagrams to generate message sequences, and the formal specification to generate state invariant predicates for the states in which each a class can receive a message. The message sequences are combined with the state invariants to construct test models, which are used to generate the test paths.

Integration testing is an important part of the overall testing process since many new types of faults can arise due to incorrect interaction of objects even though the individual classes may have been thoroughly tested. Binder identifies these integration faults as interface errors, conflicting functions, and missing functions [Bin99]. However, most of the research on formal specification based testing has focused on unit testing only [Off98] [OLAA03]. One reason for this lack of research on formal specification based integration testing is that the commonly used formal notations for object-oriented systems, such as Object-Z and VDM++, are not adapted to specifying the dynamic behavior of a system –

these notations are used to specify only the static structure of a system, i.e. classes, their attributes and operations, and relationships among classes.

The Unified Modeling Language (UML) [OMG05a] is the de facto industry standard for specifying the structure as well as behavior of object-oriented systems. Its notation is based on a set of constructs common to most object-oriented languages. However, the main difficulty in combining UML with formal methods is that the UML itself does not have formal semantics [EFLR98]. The semantics of UML have been described informally using natural language. While the Object Management Group (OMG) was responsible for standardization of the UML as a notation, the semantics of the UML is still a research issue [PB00]. Despite this limitation of the UML, the formal methods community recognizes the role of the UML in decomposing complex problems and in presenting abstract visual perspectives of the models [AS99] [ELCKAH98] [HCKKTFSMSCM95]. The use of the UML along with a formal specification offers complementary benefits such as visualization of the models, and providing higher-level structural views of the system, while the formal notations can fill in the processing details with their precise and unambiguous syntax [AS99].

The behavior of an object-oriented software system is implemented through interaction of objects. There are two complementary ways of describing this interaction of objects in UML. One is to use the UML state-charts which focus on individual objects, and the other is to use an interaction diagram which considers a collection of cooperating objects. An interaction diagram is a behavioral specification that clearly defines a sequence of

communications among cooperating objects to implement a use case functionality. There are two kinds of essentially equivalent interaction diagrams in UML, i.e., sequence diagrams and communication diagrams. In our framework, we use the latter to derive test sequences.

The collaborating objects interact with each other through message passing to implement the system behavior. The states of the objects sending and receiving a message at the time of message passing are crucial to the correct behavior of the system. The functionality provided by an object critically depends on its state, since the same object can behave differently upon receiving the same message in different states. Moreover, certain functionalities may be unavailable in certain object states, for example, the *Pop* functionality of a stack is unavailable when it is in the *empty* state.

6.1 The Proposed Approach

Our approach is based on the idea that interactions between the objects should be tested for all states of the objects involved in the interactions. We use a UML communication diagram to generate message sequences, and then use the VDM++ formal specification to derive the state invariants for the classes receiving the messages. These state invariants are partitioned using an appropriate partitioning strategy to create invariants for the sub-states. Each message sequence is then combined with sub-state invariants to construct test models. Finally, test paths are generated from the test models by under a specified coverage criterion. It is assumed that the UML model is consistent with the formal specification. Figure 6.1 gives an architectural diagram for the proposed approach. The main components of the test generation scheme are:

Message sequence generator – is responsible for generating message sequences from a communication diagram. Each communication diagram is expressed as a *message expression,* which represents the set of all valid message sequences for a communication diagram. The notation of message expressions is based on regular expressions, and is parsed by the message sequence generator to generate message sequences. Each message sequence can be represented as a *message sequence tree (MST).*

Partition analyzer – is responsible for constructing the partition predicates for the input domains of each operation. This is done using the operation pre-conditions and class invariants from the formal specification. Each predicate is transformed into a state invariant that represents an object state in which the message can be received. The predicates are derived by partitioning the input domains [Vag96] of an operation using its pre-condition and class invariant.

Test model generator – a test model is formed by combining a message sequence with the state invariants of the operations involved in the message sequence.

Test path generator – finally, the test path generator generates test paths by traversing the test model under a specified coverage criterion. A test path consists of all messages in a message sequence, with a specific sub-state invariant selected for each message.

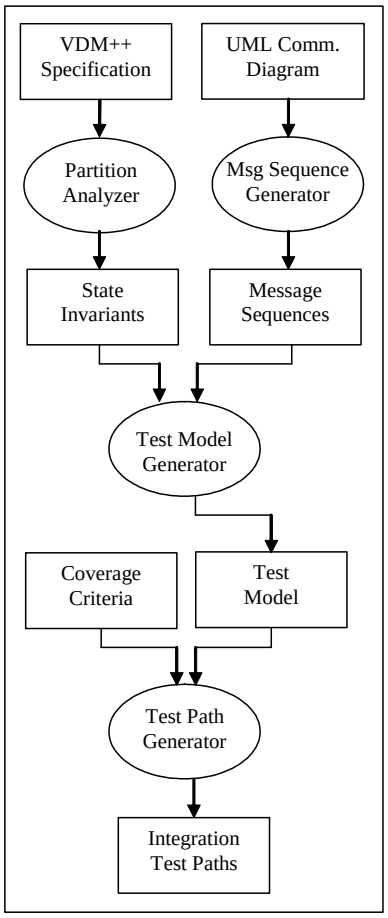

Figure 6.1. Integration testing part of the SpecTGS framework

In the following sub-sections, we explain how the proposed approach works by considering the functionality of each of these components.

6.2 Generating Message Sequences

A communication diagram in UML describes how objects collaborate with each other by message passing to provide some system functionality. A message defines a communication between a sender and a receiver object, which either causes an operation to be invoked, or an object to be created or destroyed. A message label is of the form,

[predecessor] sequence-expression message-signature

where *message-signature* consists of the *return-value*, *message-name*, and the *argument-*

list. The order of messages is determined by the *sequence-expression*, which is a hierarchical sequence number followed by an optional *recurrence expression*. The recurrence expression represents conditional or iterative execution of the message, depending on the condition specified. In iterative execution, the recurrence expression is preceded by an asterisk. The optional predecessor is a comma-separated list of sequence numbers of messages that must execute before the current message. Figure 6.2 shows an example communication diagram.

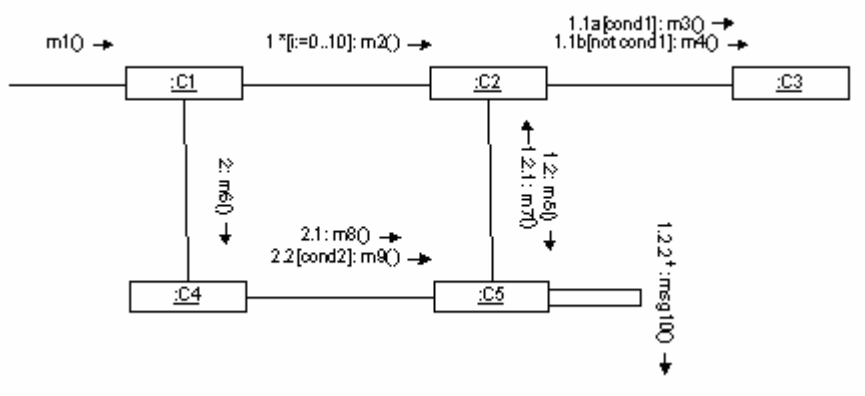

Figure 6.2. A communication diagram

A *message sequence* is a sequence of message invocations that result when a particular path is followed through a communication diagram. We define a *message expression* as an expression over the set of messages, which defines all valid message sequences for a communication diagram. Based on the notation of regular expressions, we propose a notation for *message expressions* that can be used to represent the set of all possible message sequences defined by a communication diagram. The following is a brief description of the notation:

- if m_i is a message, (m_i) denotes invocation of the message m_i
- if R_1 and R_2 are two message sub-expressions, R_1R_2 denotes sequential

execution of all message sequences derivable from R_1 followed by those derivable from R_2 at the same nesting level. For instance $(m_1)(m_2)$ denotes sequential execution of the messages m_1 and m_2

- if m_i is a message and R is a message expression, $(m_i\ R)$ denotes invocation of all message sequences derivable from the expression R from message m_i. For instance $(m_1(m_2))$ denotes invocation of message m_2 from message m_1
- $^{\wedge}R$ denotes conditional execution of message sequences derived from R
- $^{*}R$ denotes iterative execution (zero or more times) of message sequences derived from R
- $^{(m,n)}R$ denotes iterative execution of message sequences derived from R, minimum of m times and maximum of n times
- R_1+R_2 denotes mutually exclusive execution of message sequences derived from R_1 and R_2

Using this notation, a message expression for the communication diagram of Figure 6.2 can be written as:

$$R = (m_1\ ^{*}(m_2\ ((m_3) + (m_4))\ (m_5\ (m_7)\ (m_{10})))\ (m_6\ (m_8)\ ^{\wedge}(m_9)))$$

The following are some message sequences derived from the expression R:

$(m_1\ (m_6\ (m_8)))$
$(m_1\ (m_6\ (m_8)\ (m_9)))$
$(m_1\ (m_2\ (m_3)\ (m_5\ (m_7)))\ (m_6\ (m_8)\ (m_9)))$
$(m_1\ (m_2\ (m_4)\ (m_5(m_7)\ (m_{10})\ (m_{10})))\ (m_6\ (m_8)\ (m_9)))$
etc.

Each of these message sequences can be represented as a *message sequence tree (MST)* as shown in Figure 6.3.

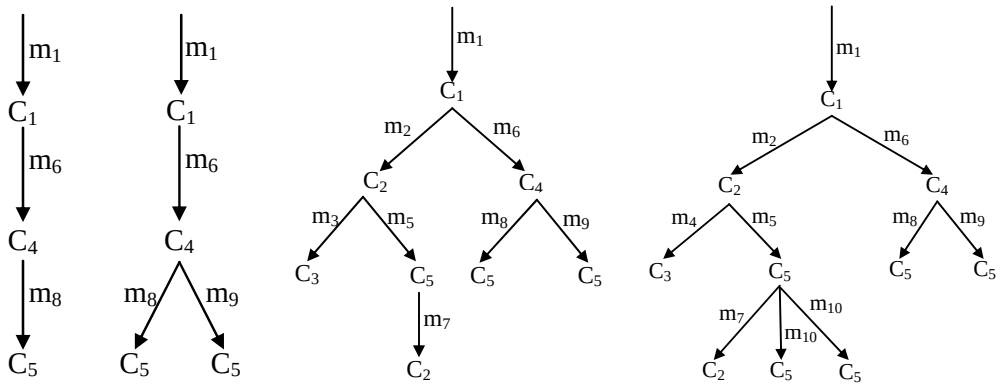

Figure 6.3. MSTs for communication diagram of Figure 6.2

In Figure 6.4, we present an algorithm to generate the set of all message sequences for a given message expression. The input to the algorithm is a message expression, and the output is the set of message sequences produced by message expression. The following is a brief explanation of the algorithm:

- if an empty message expression ε is given as input, the output is the set containing an empty message sequence ε
- if the message expression consists of a single message *(m)*, the output is the set containing *m* only, i.e., *[(m)]*
- if the message expression *R* is of the form $^{\wedge}R_1$, then the output set is the union of empty message sequence ε and the message sequences generated from R_1
- if the message expression *R* is of the form $^{*}R_1$, then the number of message sequences formed can be infinite depending on whether or not there is a bound on the number of iterations of such an iterative message. The iterative messages are repeated in message sequences according to their boundaries, as suggested by Beizer [Bei90] in the case of loop testing. For instance, if a message can execute a minimum of *m* times and a maximum of *n* times, then for the purpose

of testing, it should be executed *m* times, *n* times, and a typical value (between *m* and *n*) times. For iterative messages whose bounds are not specified, the algorithm generates message sequences containing up to three successive invocations of such messages.

- if a message expression *R* is of the form R_1+R_2, the output is the union of sets of message sequences S_1 and S_2 generated from the sub-expressions R_1 and R_2. The union function is assumed to be predefined.
- if a message expression *R* is of the form R_1R_2, where R_1 and R_2 are sub-expressions, then the output is the product of sets of message sequences S_1 and S_2 generated from R_1 and R_2 respectively. The product of two sets of message sequences S_1 and S_2 is defined as the set of all message sequences formed by concatenating message sequences of S_1 with message sequences of S_2

```
function genMsgSeqs(R : MsgExpr): set of MsgSeq
{       MSset : set of MsgSeq;
        MSset := [ ];
        if (R is ε) then MSset := [ε];
        else if (R is of the form (m)) then MSset := [ (m) ];
        else if (R is of the form ^R1) then
                MSset := union( [ε], genMsgSeqs(R) );
        else if (R is of the form R1 + R2) then
                MSset := union(genMsgSeqs(R1), genMsgSeqs(R2));
        else if (R is of the form R1 R2) then
                MSset := product(genMsgSeqs(R1), genMsgSeqs(R2));
        else if (R is of the form (m R1) ) then
                MSset := product( [ m ], genMsgSeqs(R1) );
        else if (R is of the form *R1) then
                MSset := union( [ε], genMsgSeqs(R1),
                        genMsgSeqs(R1 R1), genMsgSeqs(R1 R1 R1));
        return MSset;
}
```

Figure 6.4. Algorithm to generate message sequences

The message sequences generated in this phase are input to the test model generator.

6.3 Constructing State Invariants

Execution of a message sequence results in operation invocations on the receiving objects in a sequence. The behavior of an operation critically depends on the state of the receiving object and may also depend on states of other objects in the collaboration. Model-based formal specification notations such as VDM++, Object-Z, and the B method use predicate logic to specify class invariants and operation preconditions which implicitly define the correct state of the object in which an operation can be invoked. In Object-Z, for example, the predicate part of an operation schema defines the relationship between state variables before and after the operation. An implicit specification in VDM++, on the other hand, defines three kinds of predicates: the class invariant, the pre-condition, and the post-condition predicates. The class invariant is defined in the context of a class; the pre-condition and post-condition predicates are defined in the context of an operation.

Although the proposed approach is generic and can be applied to a variety of formal notations that are based on predicate logic, we choose VDM++ to demonstrate our approach, as it explicitly defines the pre- and post-condition predicates, as well as class invariants. In Object-Z, extraction of pre- and post-conditions from an operation schema can be difficult.

We define the *pre-state* for a message *m, prestate(m)* as the state of the receiving object in which the message *m* can be received. This state is represented as a set of all allowable

values for each state variable (or instance variables) of the receiving object, and is determined by the data types of state variables, the class invariant, and the pre-condition of the method *m.* A state invariant for this state can be derived from the conjunction of class invariant *inv(C)* of the receiving object, the pre-condition *pre(m)* of the message *m*, and the implicit *type constraints* of the state variables, i.e.,

$$E \equiv inv(C) \wedge pre(m) \wedge type\text{-}contraints$$

The type constraints are based on the declared type of a state variable, and may also arise from refinement of data types in formal specification to those in the programming language. For instance, if the VDM++ *int* type refined to *unsigned int*, or a set type of VDM++ is refined to an array in C++, it would lead to additional constraints on the values or order of elements etc. If VDM++ is used as the specification language, the predicate *prestate(m)* is a well-formed VDM++ expression that consists of one or more *clauses* joined with the *logical connectives (not, and, or),* and the *constructors* (a type of VDM++ operator used to construct the expressions). A *clause* is either a relational sub-expression, or a set membership sub-expression, or a more complex sub-expression involving operators of the types: *combinators, applicators,* and *evaluators.*

The predicate *pre(m)* in the above expression *E* may involve both state variables of class *C* and input parameters of method *m.* Thus, non-state variables are required to be eliminated from the above expression. In the following sub-section we develop a strategy to eliminate non-state variables and construct a state invariant *prestate(m)* from the expression *E*. After construction of the state invariant *prestate(m),* our goal is to construct sub-states of the state defined by *prestate(m)* which would result in generation of more

effective test cases. For this purpose, we first describe two existing strategies that have been used in partitioning of state predicates and shown to be effective in class testing, i.e. partition analysis and boundary state coverage. Our strategy is based on combination of the two strategies and using the sub-state coverage in integration testing.

6.3.1 Partition Analysis

Partition analysis is a commonly used strategy in generating partition predicates from a Boolean expression. It is based on the idea that the predicates should be tested for all possible truth assignments of clauses which make the predicate *true*. For instance, if A and B are two clauses, the expression $(A \vee B)$ can be made *true* by the following truth assignments,

$A = true \qquad B = true$
$A = true \qquad B = false$
$A = false \qquad B = true$

This is shown with a Venn diagram in Figure 6.5.

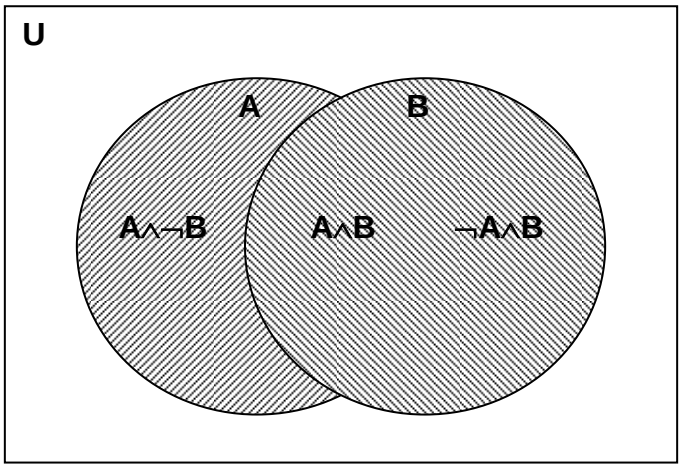

Figure 6.5: Partition analysis applied to the predicate A∨B

A systematic approach to partitioning a predicate is described in [WGS94]. Let k be the number of clauses in a predicate expression E and let the clauses be $c_1, c_2, c_3, ..., c_k$. Then the expression E can be written in *Canonical Disjunctive Normal Form (CDNF)* [WGS94], as

$$E \equiv D_1 \vee D_2 \vee D_3 \vee \ldots\ldots\ldots \vee D_n$$

where each disjunct D_i is a conjunction of the form,

$$D_i \equiv C_1 \wedge C_2 \wedge C_3 \wedge \ldots\ldots\ldots \wedge C_k$$

and $n = 2^k$ is the number of disjuncts. In the above expression, each C_i represents the clause c_i or its negation, i.e.,

$$C_i \in \{ c_i, \neg c_i \}$$

In other words, each disjunct contains exactly one occurrence of each clause. For a disjunct D_i to be *true*, each of its conjuncts C_i must be *true*. Thus each disjunct corresponds to a unique truth assignment to the clauses. The disjuncts which correspond to the truth assignments which make the original expression *false* are discarded, and the remaining disjuncts are tested. It may be noticed that the original expression contained certain non-state variables arising from the precondition expression, which have to be eliminated because they do not represent object state. The following strategy is applied to eliminate non-state variables:

- if a clause contains only non-state variables, it can be assigned the *true* value, and thus eliminated from the disjunct.
- if a clause contains both state variables and non-state variables, then boundary value analysis is applied to assign test values to the non-state variables, and the

disjunct is repeated with each test value of the non-state variables.

Each disjunct now represents a sub-state of the original state invariant for class *C* in which it can receive the message *m.*

6.3.2 Boundary State Coverage

Ambert et al. [ABCGLPVU02] introduce the concept of boundary state coverage which has been applied in BZ-TESTING-TOOLS and demonstrated by Bernard et al. [BLLP04]. The basic idea is that in a boundary state, at least one state variable has a boundary value. Kosmatov et al. [KLPU04] introduce the concept of frontiers (edges) of domains and define five different coverage criteria for boundary state coverage, i.e. One Boundary (OB) coverage, Multi-Dimensional (MD) coverage, All Edges (AE) coverage, All Edges Multi-Dimensional (AEMD) coverage, and All Boundaries (AB) coverage. However, these boundary state coverage criteria have been applied to class level testing only.

6.3.3 Partitioned Boundary State Coverage

We combine the partition analysis strategy with boundary state coverage to develop a new partitioning strategy called *partitioned boundary state coverage*. This leads to a new set of stronger coverage criteria which we apply to our integration testing approach. Figures 6.6 and 6.7 illustrate the difference between conventional boundary state coverage and partitioned boundary state coverage.

In partitioned boundary state coverage, the boundary state coverage criteria are applied to

each partition. This ensures a stronger coverage of boundary states. Depending on the boundary state coverage criterion chosen and the number of state variables, however, the number of sub-states can become exponentially large. For instance, if in a predicate expression, there are *k* clauses, *n* state variables and each variable has *b* boundary values, then,

$$\textit{Max. number of partitions} = 2^k$$
$$\textit{Number of boundary states} = b^n$$
$$\textit{Max. number of partitioned boundary states} = 2^k.b^n$$

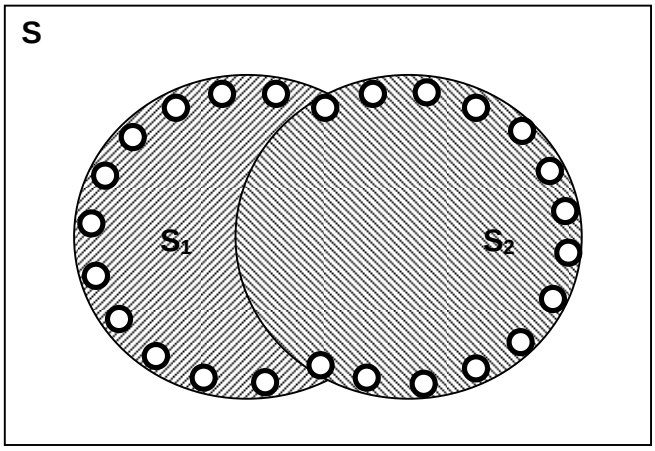

Figure 6.6: Boundary state coverage

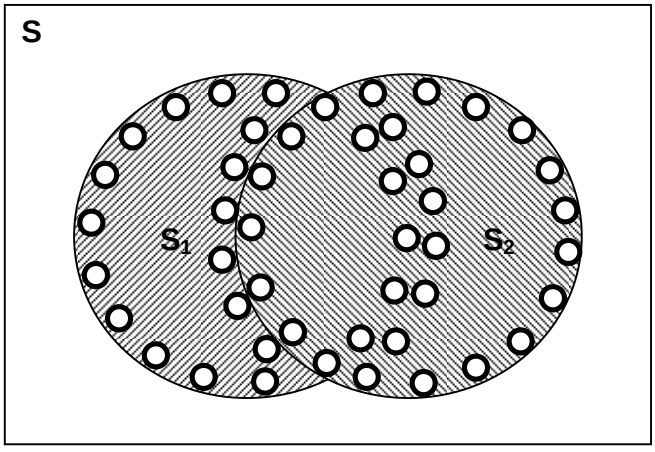

Figure 6.7: Partitioned boundary state coverage

6.3.4 Coverage Criteria for Partitioned Boundary State Testing

To ensure effective testing while reducing the number of sub-states, one can select from various coverage criteria. For this purpose, we define five coverage criteria for partitioned boundary state coverage which correspond to Kosmatov et al.'s coverage criteria [KLPU04], as follows.

Partitioned One-Boundary Coverage: This is the minimal partitioned boundary coverage criterion. It only requires that sub-states of each partition must cover at least one boundary state.

Partitioned Multi-Dimensional Coverage: This coverage criterion requires that the sub-states of each partition cover some boundary states which involve boundary values of all state variables.

Partitioned All-Edges Coverage: In Partitioned All-Edges Coverage, the sub-states of each partition must cover all edges (an *edge* is formed by a set of values of the state variables where at least one of the state variables has a boundary value).

Partitioned All-Edges Multi-Dimensional Coverage: In this coverage criterion, the sub-states of each partition must not only cover all edges, but also the boundary states which involve boundary values of all state variables.

Partitioned All-Boundaries Coverage: This is the strongest partitioned boundary

coverage criterion. It requires that sub-states of each partition cover all boundary states.

The subsumption relationships among these coverage criteria are similar to those among Kosmatov et al.'s criteria, except that these criteria are stronger and subsume partition analysis coverage as well.

6.3.5 An Example

As an example, consider a *Stack* class with the class invariant,

$$inv(Stack): \quad (top >= -1) \wedge (top < MAX)$$

and pre-condition for the *pop* operation,

$$pre(pop): \quad (top >= 0)$$

No type constraint is required if *top* is defined as of integer type in the formal specification, and is refined to the same type in the implementation. Now,

$$E \equiv (top >= -1) \wedge (top < MAX) \wedge (top >= 0)$$

This expression does not involve any clauses with non-state variables and is already in DNF with only one disjunct since there are only conjunction operators in the expression. The three conjuncts in the expression can be partitioned using boundary value analysis, as below,

$$\begin{pmatrix} top > -1 \\ top = -1 \end{pmatrix} \times \begin{pmatrix} top < MAX - 1 \\ top = MAX - 1 \end{pmatrix} \times \begin{pmatrix} top > 0 \\ top = 0 \end{pmatrix}$$

A cross product of these three sets of partitions results in 8 combinations, i.e.,

$$\begin{pmatrix} top > -1 \wedge top < MAX - 1 \wedge top > 0 \\ top > -1 \wedge top < MAX - 1 \wedge top = 0 \\ top > -1 \wedge top = MAX - 1 \wedge top > 0 \\ top > -1 \wedge top = MAX - 1 \wedge top = 0 \\ top = -1 \wedge top < MAX - 1 \wedge top > 0 \\ top = -1 \wedge top < MAX - 1 \wedge top = 0 \\ top = -1 \wedge top = MAX - 1 \wedge top > 0 \\ top = -1 \wedge top = MAX - 1 \wedge top = 0 \end{pmatrix}$$

However, only 3 of these combinations are satisfiable (assuming *MAX>1*), i.e.,

$$\begin{pmatrix} top > -1 \wedge top < MAX - 1 \wedge top > 0 \\ top > -1 \wedge top < MAX - 1 \wedge top = 0 \\ top > -1 \wedge top = MAX - 1 \wedge top > 0 \end{pmatrix}$$

These three predicates correspond to the states s_1, s_2, and s_3 in which the *Stack* object can receive the message *pop*. Thus, if the message *pop* is sent to the *Stack* class in a collaboration of objects, then it must be tested for each of the above states of the *Stack* class. We denote this set of predicates with *S(pop→Stack)*. Thus,

$$S(pop \rightarrow Stack) \equiv \begin{pmatrix} top > -1 \wedge top < MAX - 1 \wedge top > 0 \\ top > -1 \wedge top < MAX - 1 \wedge top = 0 \\ top > -1 \wedge top = MAX - 1 \wedge top > 0 \end{pmatrix}$$

This procedure is applied to each message in a message sequence, to construct the state invariants for the receiving class objects.

The partitioning of simple relational expressions, and the expressions involving finite sets, sequences, and maps is automated in *SpecTGS*. For instance, consider the following set membership expression with a universal quantifier,

forall x in set S & (x<y)

If S is a finite set of elements s_1, s_2, s_3, ……, s_n, then the above expression can be

evaluated as,

$$(s_1 < y) \text{ and } (s_2 < y) \text{ and } (s_3 < y) \text{ and } \ldots\ldots \text{ and } (s_n < y)$$

Similarly, an expression with an existential quantifier can be evaluated as,

$$(s_1 < y) \text{ or } (s_2 < y) \text{ or } (s_3 < y) \text{ or } \ldots\ldots \text{ or } (s_n < y)$$

Expressions which invoke a VDM++ function can also be partitioned automatically, provided that they do not refer to an infinite collection. This limitation is often acceptable for test generation purposes since it is common to replace an unbounded set by a small finite set of enumerated values before testing commences [LPU02a].

6.4 Constructing the Test Model

The test model is a tree structure that represents a collection of message sequences which differ only by the states of receiving objects. The messages and their order are the same in each message sequence represented by the test model. The nodes of the test model are shown as rectangular boxes that denote classes such that each class box has one or more sub-nodes, shown as bubbles, that represent states of the class corresponding to the predicates in $S(m \rightarrow C)$ where *m* is the message received by class *C*.

The test model is constructed from a message sequence tree (MST) and the state invariants for the classes involved. Consider, for example, the three states in which the *Stack* class can receive the *pop* message. The corresponding part of the test model would appear as shown in Figure 6.8.

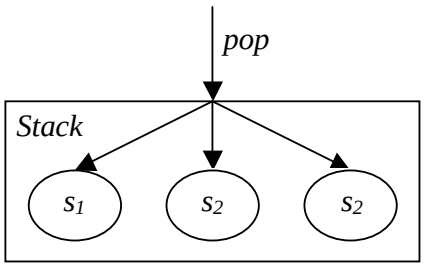

Figure 6.8. The states in which the pop message can be received

As a concrete example, consider the message sequence

$$(m_1\ (m_2\ (m_3)\ (m_5\ (m_7)))\ (m_6\ (m_8)\ (m_9)))$$

whose MST is given in Figure 6.3. Assume that s_{ijk} denotes state k of class C_i in which the message m_j can be received, where

$$k = 1, 2, 3, \ldots\ldots, n_{ij}$$

and n_{ij} = *the number of predicates in the set* $S(m_j \rightarrow C_i)$

then test model for the above message sequence would look like as shown in Figure 6.9.

6.5 Generating Test Paths

The test paths are derived by a *pre-order traversal* of the test model, such that exactly one state is selected from each class box. Thus, the number of test paths that can be derived from a test model is the product of the number of states in each class box. For instance, the number of test paths for the test model of Figure 6.9 is 1296 (i.e. 3x3x2x2x3x2x3x2). The following are some of the test paths generated from the test model of Figure 6.6,

T_1: $m_1 \rightarrow C_1{:}s_{111}$, $m_2 \rightarrow C_2{:}s_{221}$, $m_3 \rightarrow C_3{:}s_{331}$, $m_5 \rightarrow C_5{:}s_{551}$, $m_7 \rightarrow C_2{:}s_{271}$, $m_6 \rightarrow C_4{:}s_{461}$, $m_8 \rightarrow C_5{:}s_{581}$, $m_9 \rightarrow C_5{:}s_{591}$

T_2: $m_1 \rightarrow C_1{:}s_{112}$, $m_2 \rightarrow C_2{:}s_{221}$, $m_3 \rightarrow C_3{:}s_{331}$, $m_5 \rightarrow C_5{:}s_{551}$, $m_7 \rightarrow C_2{:}s_{271}$, $m_6 \rightarrow C_4{:}s_{461}$, $m_8 \rightarrow C_5{:}s_{581}$, $m_9 \rightarrow C_5{:}s_{591}$

T_3: $m_1 \rightarrow C_1{:}s_{113}$, $m_2 \rightarrow C_2{:}s_{221}$, $m_3 \rightarrow C_3{:}s_{331}$, $m_5 \rightarrow C_5{:}s_{551}$, $m_7 \rightarrow C_2{:}s_{271}$, $m_6 \rightarrow C_4{:}s_{461}$, $m_8 \rightarrow C_5{:}s_{581}$, $m_9 \rightarrow C_5{:}s_{591}$

etc.

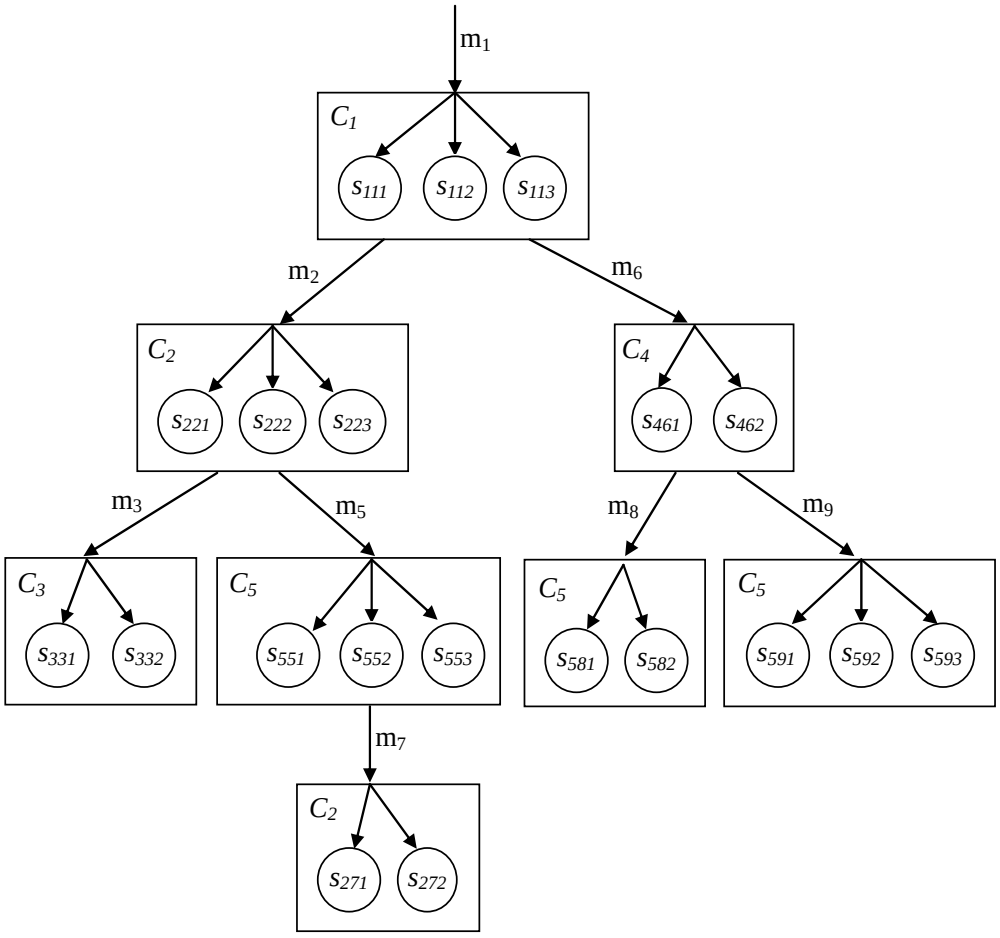

Figure 6.9. A test model for the message sequence *(m1 (m2 (m3) (m5 (m7))) (m6 (m8) (m9)))*

In the above test paths, the message $m_i \rightarrow C_j{:}s_{jik}$ denotes execution of message m_i of class C_j in state s_{jik}.

The *external message* in a communication diagram (without a sequence number) represents a *system-level operation call,* and is the first message in each test path. Once this message is invoked with appropriate input data, the rest of the messages in the test

path are automatically triggered.

6.6 Test Coverage Criteria

Since the number of test paths grows exponentially with the number of states, exhaustive coverage of all test paths can be very expensive. In this sub-section, we define various coverage criteria for the test paths.

6.6.1 Message Coverage

We define the message coverage criterion as follows: *For each message m in the communication diagram, there must be at least one test case t such that when the software is executed using t, the message m is executed at least once.* This is the minimal coverage criterion, and can be satisfied without partitioning the states. It simply requires that conditional and iterative messages be executed at least once. This can be achieved without executing all message sequences. For example, if a larger message sequence contains all messages of the communication diagram, then only one test path would be sufficient to meet message coverage.

6.6.2 Message Sequence Coverage

This coverage criterion is defined as follows: *For each message sequence s derived from a communication diagram, there must be at least one test case t such that when the software is executed using t, all messages in s are executed in the order specified by s.* This criterion requires that each message sequence be executed at least once. The number of distinct message sequences depends on the conditional and iterative messages in the communication diagram. In case of an iterative message, each message sequence contains

a distinct number of iterations, thus the number of message sequences can become quite large. In [Bei90], it is suggested that the loops should be tested at the boundaries. For instance, if an iterative message can execute a maximum of *n* times, then it should be tested by executing it *0*, *1*, *n-1*, and *n* times. This strategy can significantly reduce the number of test paths for iterative messages.

6.6.3 Message/State Coverage

This coverage criterion is defined as follows: *For each message m sent to class C, and for each state st* $\in$ *S(m*$\rightarrow$*C), there must be at least one test case t such that when the software is executed using t, the class C receives the message m in state st.* This criterion is similar to the message coverage, but it caters for each state of the message receiving class. The number of test paths in this criterion, critically depends on the number of states of recipient classes.

6.6.4 Message Sequence/State Coverage

This coverage criterion is defined as follows: *For each message sequence s derived from a communication diagram, and for each state st of a class C receiving a message m, there must be at least one test case t such that when the software is executed using t, the message sequence s is executed and the message m is received by class C in state st.* Again, this criterion is similar to the message sequence coverage, but also requires coverage of the states of classes receiving the messages. The number of generated test paths depends not only on conditional and iterative messages, but also on the number of states of each class.

6.6.5 All-Path Coverage

This is the most exhaustive coverage criterion. It requires generation of test paths with all possible combinations of states of classes involved in a message sequence. It allows testing of the methods whose functionality depends not only on the state of their own classes, but also on the states of other classes in the collaboration.

Subsumption relationships among the above-defined coverage criteria are shown in Figure 6.10.

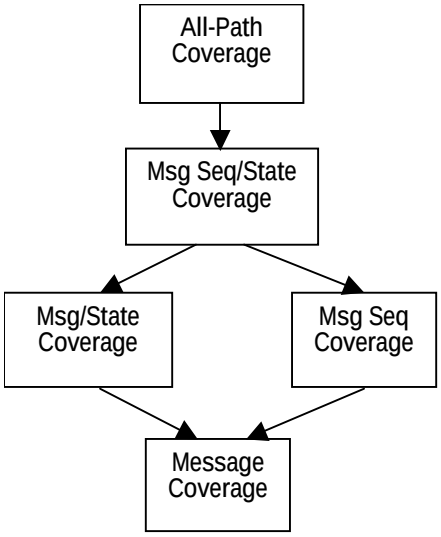

Figure 6.10. Subsumption relationships among coverage criteria

These are similar to the subsumption relationships among data flow based test criteria [FW88]. It can be seen from the figure that the message sequence/state coverage subsumes all other criteria except the all-path coverage, but as we shall see it produces a significantly lower number of test paths as compared to the all-path coverage. Thus message sequence/state coverage is an effective test coverage criterion which is also cost-effective.

As an example, let us assume that in the communication diagram of Figure 6.2, each class can receive a message in exactly *k* states, i.e.,

$$cardinality(S(m_i \rightarrow C_j)) = k, \text{ for all messages } m_i \text{ and classes } C_j$$

Then, *message coverage criterion* requires only two test paths to be executed corresponding to the two message sequences, i.e.,

$$(m_1\ (m_2\ ((m_3))\ (m_5\ (m_7)\ (m_{10})))\ (m_6\ (m_8)\ (m_9)))$$

$$(m_1\ (m_2\ ((m_4))\ (m_5\ (m_7)\ (m_{10})))\ (m_6\ (m_8)))$$

A minimum of two message sequences are required to be tested since no single message sequence covers all messages of the communication diagram due to mutual exclusiveness of the messages m_3 and m_4. It may also be noticed that the messages in above sequences may be received by the recipient classes in any states since the coverage criterion only requires the messages to be covered.

Similarly, the *message/state coverage criterion* requires that only that each message and each state be covered by the test paths. This criterion can be satisfied by *2k* test paths by generating *k* test paths from each of the above two sequences. Since each test path can cover the *i*th state (*i* varies from 1 to *k*) of each recipient class in the message sequence, therefore only *k* test paths can cover all states of the classes involved in each message sequence. Table 6.1 gives the number of test paths required to be generated for each coverage criterion applied to the example. It is obvious that the number of test paths for *all-path coverage* is much larger than those for other criteria.

Each of the above criteria requires that the data values be generated so that the coverage criterion is met. Data values are required to be generated only for the system level operation call, whose execution will automatically trigger a particular message sequence, depending on the input values and the system state. Implementation of the proposed technique also requires the code to be instrumented for setting and getting object states.

TABLE 6.1. NUMBER OF TEST PATHS AGAINST THE COVERAGE CRITERIA

Coverage Criterion	***Number of Test Paths***
Message Coverage	*2*
Msg/State Coverage	*2k*
Msg Seq Coverage[*]	$2^{n+1}+2^{n}+6$
Msg Seq/State Coverage[*]	$(2^{n+1}+2^{n}+6)k$
All-Path Coverage[*†]	$\approx k^{5n+4}$

[*] *n* is the maximum number of times that iterative messages are repeated

[†] Actual number of test paths = $k^{5n+4}+k^{5n+3}+k^{5n-1}+k^{5n-2}+k^{9}+k^{8}+k^{4}+k^{3}$

6.7 Discussion

Our integration testing approach is based on testing all message sequences resulting from collaboration of objects in a UML communication diagram. To allow more effective testing, we construct sub-states from the state invariant for each message receiving class in the collaboration. For this purpose, we combine the traditional partition analysis strategy, commonly used in formal specification based class testing, with boundary state coverage strategy which has been successfully applied in the context of class testing. The complete coverage of a predicate is similar to path coverage of the code which results in combinatorial explosion. To control this combinatorial explosion, the sub-state predicates are constructed based on the well-known testing heuristics of equivalence class

partitioning and boundary value analysis. The testing literature shows that testing at boundary values is more likely to reveal bugs due incorrect implementation of conditions. Thus, our new strategy results in stronger coverage of the state predicates which leads to more effective testing. Also, it has been empirically shown in [ABRAZN06], that integration testing strategy based on states of classes involved in a collaboration results in more effective test cases which are able to detect faults occurring due to invalid states of objects. In particular, the authors used mutation operators WIS (Wrong Initial State), TSSS (Target State as Source State), and WCS (Wrong Calling State) to seed state-dependent faults which were caught by the state based integration testing strategy.

We have also defined various coverage criteria for partitioned boundary state coverage based on Kosmatov et al.'s coverage criteria, and new coverage criteria for test path generation. These coverage criteria can be used to measure adequacy of an existing set of test cases, or to control the process of test generation. The *SpecTGS* uses these criteria for the latter purpose – the tester can choose the appropriate criteria to generate test cases.

Chapter 7

SpecTGS Implementation and Evaluation

This chapter covers a brief description of the prototype tool that implements our proposed framework, and a detailed case study that demonstrates the proposed integration testing approach. The case study is a railway Control Speed Limitation and Monitoring (CSLaM) system adapted from [FLMP05].

7.1 Implementation

A proof-of-concept, prototype tool for the proposed framework *SpecTGS* has been implemented in Java language [MMA06]. The prototype tool consists of two main components, i.e., for unit testing and integration testing. A high-level architectural diagram of the tool is shown in Figure 7.1.

The unit testing component of the tool accepts as its input a text file containing VDM++ specification of the class to be tested, and another text file containing a C++ implementation of the class. The user selects from single operation testing and operation

sequences testing. In the case of single operation testing, the tool constructs method predicate for the selected operation, partitions it using canonical DNF form, eliminates unsatisfiable partitions, generates test data using boundary value analysis, and constructs concrete executable test cases as output. If operation sequence testing is selected by the user, the tool parses the trace structure of the class specification and generates valid operation sequences, then for each method in an operation sequence, it constructs the method predicate, converts it to the canonical DNF and generates test data using boundary value analysis on each partition.

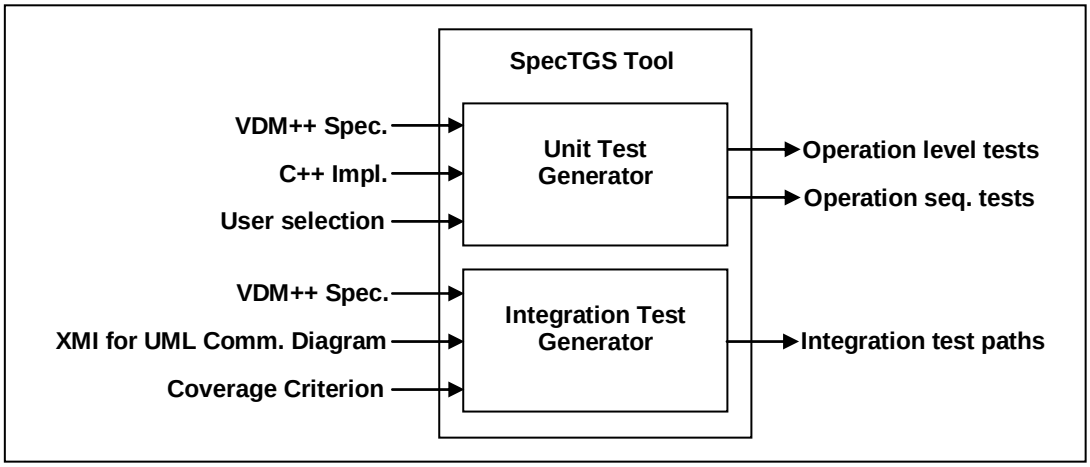

Figure 7.1. Architecture of the SpecTGS Tool

The integration testing component of the tool requires a VDM++ specification and a UML communication diagram to generate test paths. The tool accepts UML communication diagram in XMI (XML Metadata Interchange) format. XMI is an OMG standard [OMG05b] for defining, interchanging, and manipulating UML artifacts using XML format. The common UML diagramming tools such as the Borland's Together and Rational Rose allow UML diagrams to be exported in XMI format. The tool accepts the

VDM++ specification as a text file. It is assumed that the VDM++ specification is consistent with the UML communication diagram. The tool does not perform any consistency analysis between the two inputs. A message expression is then constructed for the communication diagram, as described in chapter 6, and message sequences are generated from the expression. State invariants for the states in which a class can receive a message are constructed from the VDM++ specification using partition analysis, and a test model is constructed for each message sequence. Finally, test paths are generated according to the coverage criterion specified by the user. We have tested the *SpecTGS* tool on several examples, and the results show that it effectively generates test cases under the specified criteria.

7.2 Case Study

In this section, we present a railway Control Speed Limitation and Monitoring (*CSLaM*) system, adapted from [FLMPV05], as a case study to demonstrate the proposed testing strategy. The purpose of the *CSLaM* system is to continuously monitor the train speed, and activate the emergency brake if the train's speed is above a threshold value. The threshold value is computed as the sum of the maximum permitted speed a small constant such as 5 or 10. The system is intended to be used in situations where speed of the train is required to be controlled in certain areas, e.g., when repairs are taking place along a section of the track. The speed restrictions are signaled by different types of beacons placed along side the track. The maximum permitted speed is determined as the minimum of the maximum speed of which the train is capable (e.g. 180 km/h), and the speed limit imposed by speed restriction beacons. Obviously, this is a safety-critical system, because failure of the system could lead to an accident which, in turn, could result in loss of

human lives.

The *CSLaM* system consists of an on-board control speed limitation (*CSL*) subsystem and the trackside beacons. The *CSL* subsystem is further composed of an on-board computer, a cab display, and an emergency brake. The cab display contains three lighting indicators – an alarm indicator, an emergency brake indicator, and a ground fault indicator. The alarm indicator is turned on when train speed exceeds the *alarm speed* and the other two indicators are off. Alarm speed is obtained by adding a constant (e.g. 5 km/h) to the maximum permitted speed. The emergency brake indicator turns on when the train speed exceeds *emergency brake speed*. The emergency brake speed is obtained by adding another constant (e.g. 10 km/h) to the maximum permitted speed, such that emergency brake speed is greater than the alarm speed. The on-board computer is responsible for checking if the speed of the train is within the allowed limit, or has exceeded the alarm speed, or has exceeded the emergency brake speed. The emergency brake can be set by the *CSL* subsystem if train speed exceeds the emergency brake speed.

There are four types of beacons that can be encountered by the train alongside the track, i.e.,

- An *Announcement Beacon* announces the arrival of a *Limitation Beacon*. The information provided by an announcement beacon is a speed limit which must be respected when a limitation beacon is reached.
- A *Limitation Beacon* enforces the speed restriction as soon as head of the train meets it. The speed restriction remains into effect until tail of the train meets an *End Beacon*. If a limitation beacon is not preceded by an announcement

beacon, a ground fault is raised.

- A *Cancel Beacon* cancels all announcements made. The cancel beacon is ignored if no announcement is present.
- An *End Beacon* marks the end of a speed limitation area. The train returns to its normal speed when its tail meets an end beacon.

7.2.1 UML Models

A class diagram for the *CSLaM* system is given in Figure 7.1, and a detailed VDM++ specification for the *CSLaM* system is given in Appendix-II. The following external events have been identified, which trigger the corresponding operations in the CSLaM system:

- *HeadMeetsBeacon:* This event occurs when head of the train encounters a beacon. Depending on the type of beacon met, appropriate action is taken by the system. For instance, when head of the train meets a LimitBeacon, a new speed restriction comes into force.
- *TailMeetsBeacon:* This event occurs when tail of the train meets a beacon. Again appropriate action is taken depending on the type of beacon met.
- *NoBeaconMet:* This event occurs when an AnnounceBeacon is met, but a corresponding LimitBeacon is not met after the specified distance. A ground fault is raised by the system in such a case.
- *CheckSpeed:* This event occurs automatically at fixed time intervals to allow for continuous monitoring and control of the train speed.

Figure 7.2 shows a UML communication diagram for the *HeadMeetsBeacon* event. XMI output generated by Borland's Together for communication diagram of Figure 7.2 is given in Appendix-III.

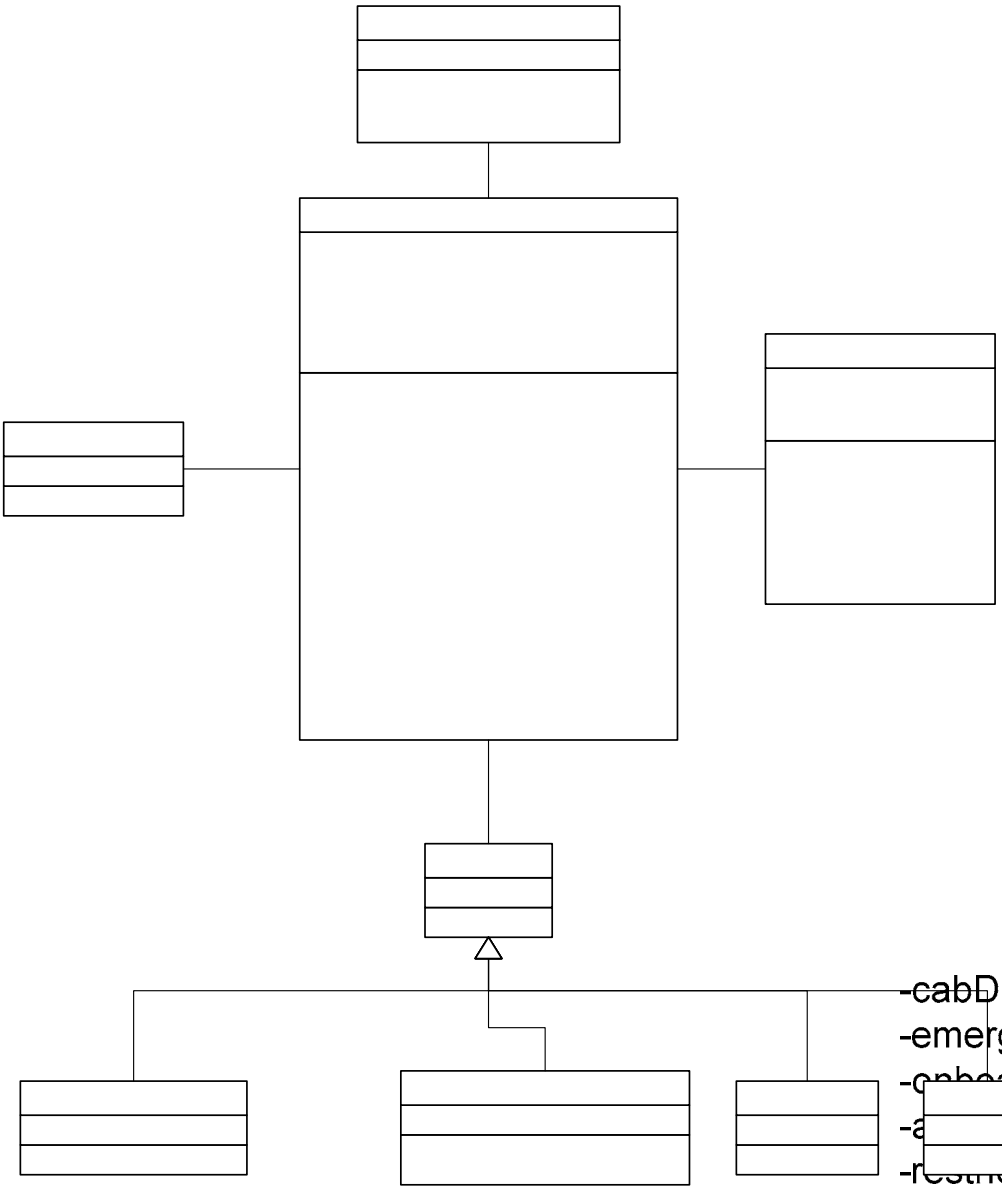

Figure 7.1: Class diagram for CSLaM system

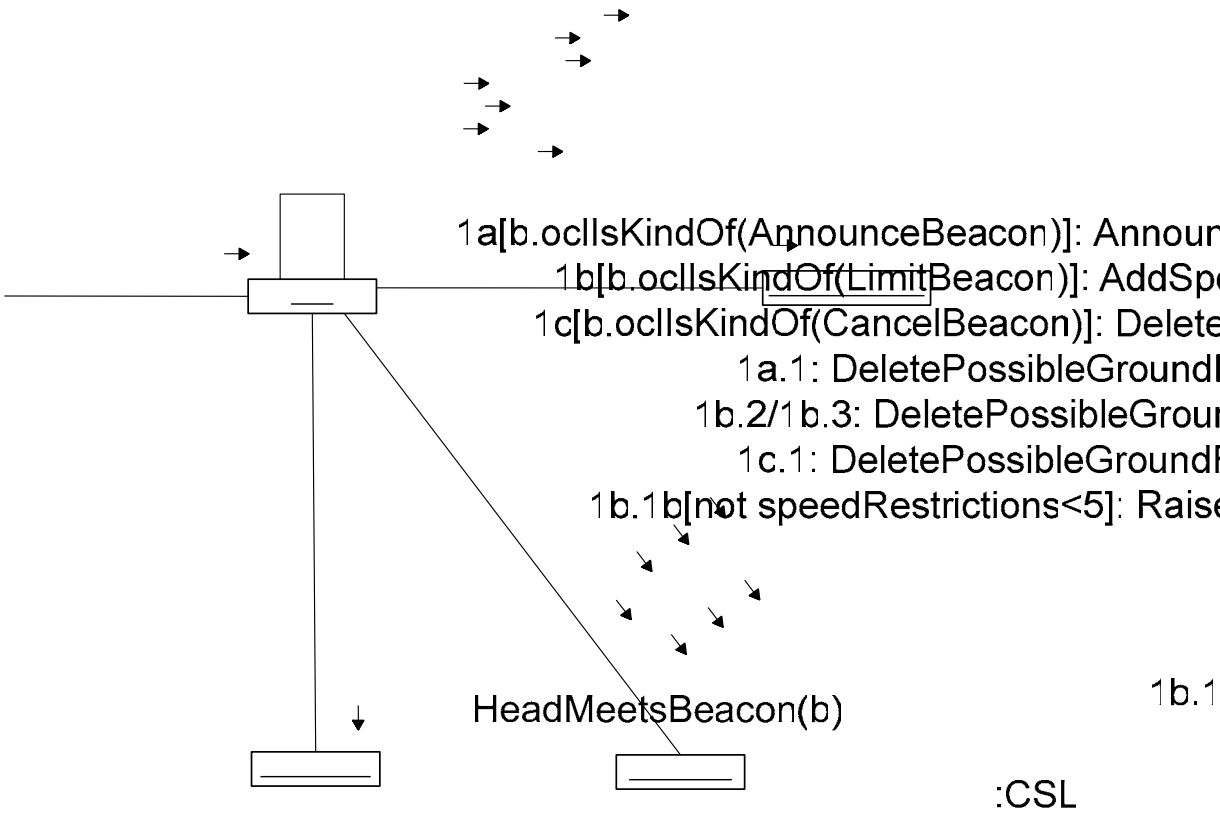

Figure 7.2: Communication diagram for the *HeadMeetsBeacon* event

7.2.2 Generating Message Sequences

A message expression for the communication diagram of Figure 7.2 is computed as:

$$(m_1((m_2(m_5(m_{11})\hat{}(m_{12}))) + (m_3((m_6)(m_7)(m_8(m_{13})\hat{}(m_{14})) + (m_9(m_{15})))) + (m_4(m_{10}(m_{16})\hat{}(m_{17}))))$$

where, messages have been represented as:

m_1 = *HeadMeetsBeacon*
m_2 = *AnnounceSpeedRestriction*
m_3 = *AddSpeedRestriction*
m_4 = *DeleteAnnouncements*
m_5 = *DeletePossibleGroundFault*
m_6 = *GetTartgetSpeed*
m_7 = *SetSpeedRestriction*
m_8 = *DeletePossibleGroundFault*
m_9 = *RaiseGroundFault*
m_{10} = *DeletePossibleGroundFault*
m_{11} = *GetDisplay*
m_{12} = *UnsetGroundFault*
m_{13} = *GetDisplay*
m_{14} = *UnsetGroundFault*
m_{15} = *SetGroundFault*
m_{16} = *GetDisplay*
m_{17} = *UnsetGroundFault*

Since there are no iterations in the message expression, the number of message sequences generated would be finite. By applying the algorithm for message sequence generation, the following seven message sequences can be generated from the above message expression, i.e.,

s_1: $(m_1(m_2(m_5(m_{11}))))$

s_2: $(m_1(m_2(m_5(m_{11})(m_{12}))))$

s_3: $(m_1(m_3(m_6)(m_7)(m_8(m_{13}))))$

s_4: $(m_1(m_3(m_6)(m_7)(m_8(m_{13})(m_{14}))))$

s_5: $(m_1(m_3(m_9(m_{15}))))$

s_6: $(m_1(m_4(m_{10}(m_{16}))))$

s_7: $(m_1(m_4(m_{10}(m_{16})(m_{17}))))$

Each of these message sequences can be represented as a message sequence tree (MST) as shown in Figure 7.3.

7.2.3 Constructing State Invariants

Let us consider the first message sequence s_1. It contains four messages, i.e., m_1, m_2, m_5, and m_{11}. Now, message m1 is received by class CSL, thus prestate for this message $m_1 \rightarrow$ *CSL*, can be determined by conjoining the class invariant of CSL and the precondition of operation *HeadMeetsBeacon*, i.e., *prestate*($m_1 \rightarrow$*CSL)* is,

```
(isofclass(LimitBeacon,b)=>(len announcements>0)) and
(len speedRestrictions<=5)
```

The above expression in DNF is,

```
(isofclass(LimitBeacon,b) and (len announcements>0) and
(len speedRestrictions<=5)) or (not isofclass(LimitBeacon,b)
 and (len announcements>0) and (len speedRestrictions<=5)) or
(not isofclass(LimitBeacon,b) and not (len announcements>0) and
(len speedRestrictions<=5))
```

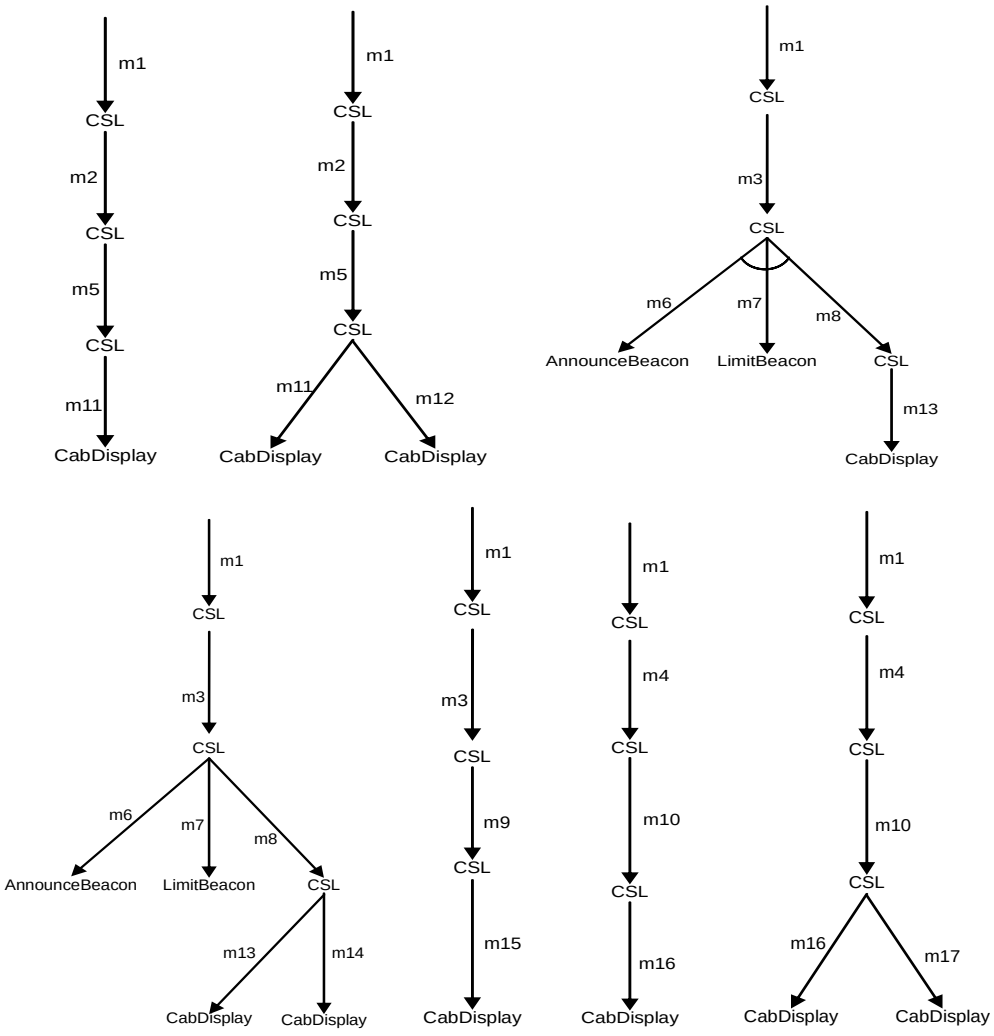

Figure 7.3: MSTs for the HeadMeetsBeacon event

Eliminating the clause `(isofclass(LimitBeacon,b))` which involves non-state variables, and simplifying the expression, we get,

```
((len announcements>0) and (len speedRestrictions<=5)) or
 (not (len announcements>0) and (len speedRestrictions<=5))
```

Thus, the class *CSL* must be in a state defined by the above expression, when the message *HeadMeetsBeacon* is received. Applying boundary value analysis on each disjunct, we get,

$$\begin{pmatrix} len\ announ\ \ cements\ \ > 1 \\ len\ announ\ \ cements\ \ = 1 \end{pmatrix} \times \begin{pmatrix} len\ speedR\ \ estriction\ \ s) = 5 \\ len\ speedR\ \ estriction\ \ s) < 5 \end{pmatrix}$$

and $$\begin{pmatrix} len\ announ\ cements\ \ = 0 \\ len\ announ\ cements\ \ < 0 \end{pmatrix} \times \begin{pmatrix} len\ speedR\ estriction\ s) = 5 \\ len\ speedR\ estriction\ s) < 5 \end{pmatrix}$$

which results in 8 combinations, of which only the following three are satisfiable,

$$len\ announcements > 1\ and\ len\ speedRestrictions = 5$$
$$len\ announcements > 1\ and\ len\ speedRestrictions < 5$$
$$len\ announcements = 1\ and\ len\ speedRestrictions < 5$$

Thus the class CSL can receive the message *HeadMeetsBeacon* in one of the three states represented by these state invariants. Let us label the three state invariants as p_1, p_2, and p_3, i.e.,

p_1 ≡ `len announcements > 1 and len speedRestrictions = 5`
p_2 ≡ `len announcements > 1 and len speedRestrictions < 5`
p_3 ≡ `len announcements = 1 and len speedRestrictions < 5`

Likewise, the state invariants of the receiving classes for other messages can be computed and labeled. Table 7.1 gives the number of states for each class in the communication diagram, for each message that it can receive. The states have been computed using the technique described in chapter 6.

7.2.4 Constructing Test Model

By combining the message sequence and the states of receiving classes, a test model can be constructed for sequence s_1 as shown in Figure 7.4.

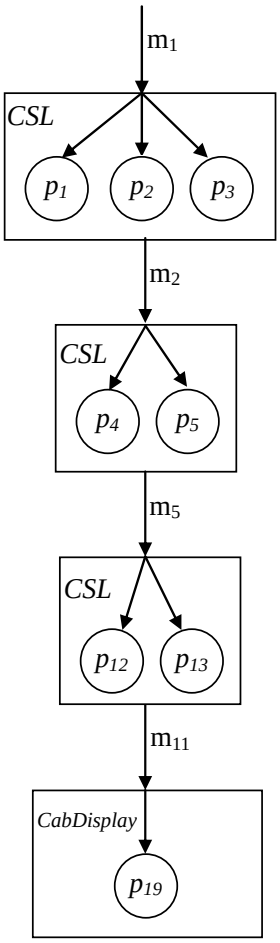

Figure 7.4: Test model for the message sequence ($m_1(m_2(m_5(m_{11})))$)

7.2.5 Generating Test Paths

The all-path coverage criterion applied to the test model of Figure 7.4 gives a total of 3x2x2x1=12 test paths, i.e.,

T_1: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_2: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_3: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_4: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_5: $(m_1 \rightarrow CSL{:}p_2(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_6: $(m_1 \rightarrow CSL{:}p_2(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_7: (m_1 →CSL:p_2(m_2 →CSL:p_5(m_5 →CSL:p_{12}(m_{11} →CabDisplay:p_{19}))))

T_8: (m_1 →CSL:p_2(m_2 →CSL:p_5(m_5 →CSL:p_{13}(m_{11} →CabDisplay:p_{19}))))

T_9: (m_1 →CSL:p_3(m_2 →CSL:p_4(m_5 →CSL:p_{12}(m_{11} →CabDisplay:p_{19}))))

T_{10}: (m_1 →CSL:p_3(m_2 →CSL:p_4(m_5 →CSL:p_{13}(m_{11} →CabDisplay:p_{19}))))

T_{11}: (m_1 →CSL:p_3(m_2 →CSL:p_5(m_5 →CSL:p_{12}(m_{11} →CabDisplay:p_{19}))))

T_{12}: (m_1 →CSL:p_3(m_2 →CSL:p_5(m_5 →CSL:p_{13}(m_{11} →CabDisplay:p_{19}))))

Similarly, test paths can be computed for all message sequences using the *number of states* information given in table 7.1. Table 7.2 gives the total number of test paths for each message sequence. A total of 120 test paths are generated for the *HeadMeetsBeacon* event under the all-path coverage criterion. A complete list of test paths for the *HeadMeetsBeacon* event is given in Appendix-IV.

TABLE 7.1: NUMBER OF RECEIVING CLASS STATES FOR EACH MESSAGE IN COMMUNICATION DIAGRAM FOR *HEADMEETSBEACON* EVENT

Message	**Received by (Class)**	**No. of states**	**State Labels**
m_1	*CSL*	*3*	*p1, p2, p3*
m_2	*CSL*	*2*	*p4, p5*
m_3	*CSL*	*4*	*p6, p7, p8, p9*
m_4	*CSL*	*2*	*p10, p11*
m_5	*CSL*	*2*	*p12, p13*
m_6	*AnnounceBeacon*	*1*	*p14*
m_7	*LimitBeacon*	*1*	*p15*
m_8	*CSL*	*2*	*p16, p17*
m_9	*CSL*	*2*	*p18, p13*
m_{10}	*CSL*	*2*	*p12, p13*
m_{11}	*CabDisplay*	*1*	*p19*
m_{12}	*CabDisplay*	*1*	*p19*
m_{13}	*CabDisplay*	*1*	*p19*
m_{14}	*CabDisplay*	*1*	*p19*
m_{15}	*CabDisplay*	*1*	*p19*
m_{16}	*CabDisplay*	*1*	*p19*
m_{17}	*CabDisplay*	*1*	*p19*

TABLE 7.2: NUMBER OF TEST PATHS GENERATED FOR EACH MESSAGE SEQUENCE OF *HEADMEETSBEACON* EVENT

Message sequence	No. of test paths
s_1	*12*
s_2	*12*
s_3	*24*
s_4	*24*
s_5	*24*
s_6	*12*
s_7	*12*
Total	*120*

Similar results for other coverage criteria and events are presented in Table 7.3.

TABLE 7.3: NUMBER OF TEST PATHS AGAINST COVERAGE CRITERIA FOR THE *HEADMEETSBEACON* EVENT

Coverage Criterion	Total
Message coverage	*4*
Message sequence coverage	*7*
Message/State coverage	*14*
Message sequence/State coverage	*24*
All-path coverage	*120*

As it can be seen from table 7.3, the number of test paths against All-path coverage is five times the next-to-complete criterion, i.e. Message sequence/State coverage criterion. Since Message sequence/State coverage criterion provides complete coverage of all message sequences as well as all states of receiving classes with minimal number of test paths, we conclude that in cost-constrained situations, this criterion may be used without losing too much testing effectiveness. In particular, as shown in [ABRAZN06], only the faults due to incorrect states of calling objects, can escape with this coverage criterion.

7.3 Evaluation of the *SpecTGS* Framework

In this section, the *SpecTGS* framework is evaluated using the evaluation criteria defined in Chapter 3. The following is a brief analysis of the *SpecTGS*:

- *Object-orientation:* The proposed framework is based on VDM++ formal specification language and UML communication diagrams, thus it fully supports the object-oriented paradigm.
- *Testing level:* The *SpecTGS* framework generates both unit level (class level) and integration level test cases. In chapter 5, it was shown that the generated test cases may also be used for inheritance and polymorphic testing.
- *Strategy Flexibility:* The framework uses the novel partitioned boundary state coverage strategy for generation of integration test paths. However, the framework can be configured to use other black-box testing strategies such as partition analysis, boundary state coverage and predicate coverage.
- *Notation Adaptability:* The class testing part of the proposed framework can be easily adapted to other model-based formal specification languages such as Object-Z, however, it would require allowable operation sequences for a class to be formally specified as proposed in [NR05]. The integration testing part of the framework not only requires the VDM++ specification, but also corresponding UML communication diagrams for extraction of message sequences. Thus, it cannot be easily adapted to other formal notations.
- *Automation and Tool Support:* The proposed technique is highly automatable – the only limitation is that test data generation for integration test paths is not yet automated. A prototype tool has also been developed to support much of the test

generation process. The tool has been used on some practical examples, including a case study described in this chapter.

Chapter 8

Conclusion and Future Work

The industrial use of formal methods is rapidly growing with the increasing role of software in safety-critical systems. The application of formal methods in specification phase eliminates ambiguities and inconsistencies in the specification and leads to fewer bugs in design and coding phases. However, a formal specification does not eliminate the need for testing. Specification based testing is done to ensure that the implementation conforms to the specification. Manual testing is a tedious, error-prone and costly activity. The formal specification can be used as a basis for automatic generation of test cases. The next section concludes the work presented in this thesis.

8.1 Conclusion

In existing formal specification based testing techniques, the focus has been on unit-level (or class-level) testing only. This limitation is due to the fact that the existing formal notations do not support specification of dynamic system behavior. In object-oriented systems, the dynamic behavior of the system must be specified in terms of interactions between the objects in order to support generation of integration-level test cases. UML is a set of diagramming notations used widely in the industry for specification of behavior

of the object-oriented systems. The use of UML along with a formal specification offers complementary benefits such as visualization of the system behavior while retaining the precise and unambiguous system specification.

The *SpecTGS* framework automates the generation of test cases for an object-oriented system from a VDM++ formal specification and UML communication diagrams. It generates unit level test cases from the VDM++ specification of the system, while integration level test cases are generated by combining information from the VDM++ specification and corresponding UML communication diagrams.

The major contribution of this thesis is to combine the VDM++ formal specification with UML communication diagrams to generate integration level test cases. The UML communication diagrams are used to construct allowable message sequences, while the formal specification is used to construct state invariants for the states in which a class can receive a message. This requires partitioning of the predicate representing the state of the class when a message is received. We proposed a new strategy for partitioning of the state predicate, called *partitioned boundary state coverage* which combines two existing strategies, i.e., partition analysis and boundary state coverage, and allows more thorough testing due to more specific state invariants. This strategy was employed in the integration testing part of the framework. The test model constructed by combining information from UML communication diagrams and the formal specification is traversed under a selected coverage criterion to generate integration test paths.

It is thus concluded that the *SpecTGS* framework's contribution is an improvement upon the state-of-the-art in formal specification based software testing. Further, it is expected that the *SpecTGS* will be beneficial for testing of safety-critical systems.

8.2 Future Directions

The future work for the *SpecTGS* framework is to evaluate its effectiveness on more large scale real-life case studies. Another direction for the future work is to automate the generation of test data for integration testing.

The OMG has not yet defined formal semantics for UML diagrams. Presently, research is under way on formalization of UML diagrams. When this happens, one of the future goals would be to exploit the testing information from the formalized UML models. Another possibility is to extend the formal notation itself to allow the specification of class interactions.

References

[AA92] N. Amla, P. Ammann, "Using Z Specifications in Category Partition Testing," *Proceedings of the Seventh Annual Conference on Computer Assurance (COMPASS '92),* June 1992, IEEE Computer Society Press, 1992.

[ABCGLPVU02] F. Ambert, F. Bouquet, S. Chemin, S. Guenaud, B. Legeard, F. Peureux, N. Vacelet, M. Utting, "BZ-TT: A Tool-set for Test Generation from Z and B using Constraint Logic Programming," *Proceedings of the CONCUR'02 Workshop on Formal Approaches to Testing of Software (FATES'02),* pages 105-120, Brnö, Czech Republic, August 2002.

[ABRAZN06] S. Ali, L.C. Briand, M.J. Rehman, H.B. Asghar, Z. Zafar, A. Nadeem, "A State Based Approach to Integration Testing for Object-Oriented Programs," accepted for publication in the *Journal of Information and Software Technology,* Elsevier Science, 2006.

[AO00] A. Abdurazik and J. Offutt, "Using UML collaboration diagrams for static checking and test generation," *The Third International Conference on the Unified Modeling Language (UML '00),* pp. 383-395, York, UK, October 2000.

[AS99] S. Agerholm and W. Schafer, *"Analyzing SAFER using UML and VDM++,"* in J. Fitzgerald, P.G. Larsen, editors, VDM in Practice, pp.139-141, September 1999.

[Att00] R. Atterer, *Automatic Test Data Generation from VDM-SL Specifications,* Diploma dissertation, The Queens University of Belfast, April 2000.

[BB96] M. Blackburn, R. Busser, "TVEC – A tool for developing critical systems," Proceedings of the Annual Conference on Computer Assurance (COMPASS'96), IEEE Computer Society Press, 1996.

[BB00] F. Basanieri and A. Bertolino, “A Practical Approach to UML-Based Derivation of Integration Tests,” *Proceedings of Software Quality Week Europe*.

[BBM01] F. Basanieri, A. Bertolino, and E. Marchetti, “COWTest: Cost Weighted Test Strategy,” *Proceedings of ESCOM-SCOPE 2001,* London, England.

[Bei90] B. Beizer, “*Software Testing Techniques,”* 2nd Edition, Van Nastrand Ranhald, 1990.

[Bei95] B. Beizer, “*Black-Box Testing: Techniques for Functional Testing of Software and Systems,”* John Wiley & Sons Inc., 1995, ISBN 0-471-12094-4.

[BH95] J.P. Bowen and M.G. Hinchey, “Ten Commandments of Formal Methods,” *IEEE Computer,* April 1995.

[Bin99] R.V. Binder, “*Testing Object-Oriented Systems: Models, Patterns and Tools,”* Addison-Wesley Object Technology Series, 1999.

[BKM02] C. Boyapati, S. Khurshid, and D. Marinov, “Korat: Automated Testing Based on Java Predicates,” *ACM ISSTA 2002.*

[BLLP04] E. Bernard, B. Legeard, X. Luck, and F. Peureux, “Generation of Test Sequences from Formal Specifications: GSM 11-11 Standard case study,” *The Journal of Software Practice and Experience,* Wiley-InterScience, 2004.

[CMMMS00] D. Carrington, I. MacColl, J. McDonald, L. Murray, and P. Strooper, “From Object-Z Specifications to Classbench Test Suites,” *Journal of Software Testing, Verification and Reliability,* Vol. 10, No. 2, pp. 111-137, 2000.

[CS94] D. Carrington, and P. Stocks, “A Tale of Two Paradigms: Formal Methods and Software Testing,” *ZUM ’94, Z User Workshop,* Springer-Verlag, pp. 51-68, 1994.

[CSK05] CSK Corporation, *VDMTools, The VDM++ Language,* version 6.8.1, 2005, http://www.csk.co.jp/support_e/vdm/index.html.

[CW96] E.M. Clarke, J.M. Wing, et. al., "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys,* Vol. 28, No. 4, December 1996.

[Daw91] J. Dawes, *The VDM-SL Reference Guide*, Pitman, London, 1991.

[DDH72] O.J. Dahl, E.W. Dijkstra, C.A.R. Hoare, "Structured Programming," *APIC Studies in Data Processing, number 8,* Academic Press, 1972.

[DF91] R.K. Doong, P.G. Frankl, "Case Studies on Testing Object-Oriented Programs," *Proceedings of the Fourth Symposium on Software Testing, Analysis and Verification,* Victoria, British Columbia, Canada, IEEE Computer Society Press, October 1991.

[DF93] J. Dick, and A. Faivre, "Automating the Generation and Sequencing of Test Cases from Model-based Specifications," *Proceedings of FME '93: Industrial-Strength Formal Methods,* Pages 268-284, Odense, Denmark, 1993, Springer-Verlag.

[DK92] Eugène Dürr, Jan van Katwijk, "VDM++, A Formal Specification Language For Object-Oriented Designs," IEEE, 1992.

[EFLR98] A. Evans, R. France, K. Lano, and B. Rumpe, "The UML as a Formal Modeling Notation," *The Unified Modeling Language Workshop (UML '98) Proceedings,* (Jean Bezivin and Pierre-Allaine Muller eds.), Springer-Verlag, LNCS 1618, 1998.

[ELCKAH98] S. Easterbrook, R.R. Lutz, R. Covington, J.C. Kelly, Y. Ampo, and D. Hamilton, "Experiences Using Lightweight Formal Methods for Requirements Modeling," *IEEE Transactions on Software Engineering,* 24(1):1-11, January 1998.

[Elm73] W.R. Elmendorf, "Cause-Effect Graphs in Functional Testing," *Technical Report TR-00.2487,* IBM Systems Development Division, Poughkeepsie, New York, 1973.

[FD96] A. Finkelstein, J. Dowell, "A comedy of errors: the London Ambulance Service case study," *Proceedings of the 8th International Workshop on Software Specification and Design,* IEEE Computer Society, 1996.

[FL02] F. Fraikin and T. Leonhardt, “SeDiTeC — Testing Based on Sequence Diagrams”, *17th IEEE International Conference on Automated Software Engineering (ASE'02),* pp. 261-266.

[FLMPV05] J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object-oriented Systems,* Springer-Verlag, 2005, ISBN 1-85233-881-4.

[FW88] P.G. Frankl and E.J. Weyuker, “An Applicable Family of Data Flow Testing Criteria,” *IEEE Transactions on Software Engineering,* 14(10), October 1988.

[GHW85] J. Guttag, J. Horning, J. Wing, “The Larch Family of Specification Languages,” *IEEE Transactions on Software Engineering,* September 1985, pp. 24-36.

[GOC06] L. Gallagher, J. Offutt, and A. Cincotta, “Integration testing of object-oriented components using finite-state machines,” to appear in the *Journal of Software Testing, Verification, and Reliability,* 2006, published online 12 Jan. 2006, DOI 10.1002/stvr.340.

[Hal88] P.A.V. Hall, “Towards Testing with respect to Formal Specification,” *Proceedings of the Second IEE/BCS Conference on Software Engineering,* 1988, pp. 159-163, IEE, 1988.

[HCKKTFSMSCM95] D. Hamilton, R. Covington, J. Kelly, C. Kirkwood, M. Thomas, A.R. Flora-Holmquist, M.G. Staskauskas, S.P. Miller, M. Srivas, G. Cleland, and D. MacKenzie, “Experiences in Applying Formal Methods to the Analysis of Software and System Requirements,” *Proceedings of the 1st Workshop on Industrial-Strength Formal Specification Techniques,* pages 30-43, IEEE Computer Society Press, April 1995.

[Hie97] R.M. Hierons, “Testing from a Z Specification,” *Journal of Software Testing, Verification and Reliability,* 1997.

[HKC95] H.S. Hong, Y.R. Kwon, and S.D. Cha, “Testing of Object-Oriented Programs Based on Finite State Machine,” *Proceedings of the Asia-Pacific Software Engineering Conference,* pp. 234-241, 1995.

[HNS97] S. Helke, T. Neustupny, and T. Santen, "Automating Test Case Generation from Z Specifications with Isabelle," *Proceedings of the 10th International Conference of Z Users*, 1997, Springer-Verlag.

[Jon90] C.B. Jones, "*Systematic Software Development Using VDM,"* Second Edition, Series in Computer Science, Prentice-Hall, New Jersey, 1990.

[KLPU04] N. Kosmatov, B. Legeard, F. Peureux, M. Utting, "Boundary Coverage Criteria for Test Generation from Formal Models," 2004.

[KT98] James D. Kiper, James E. Tomayko, "Techniques for Safety Critical Software Development," *IEEE*, 1998.

[Lay93] G.T. Laycock, "The Theory and Practice of Specification Based Software Testing," *PhD Thesis*, Department of Computer Science, University of Sheffield, 1993.

[LH85] D.C. Luckham, F.W. von Henke, "An Overview of Anna: A Specification Language for Ada," *IEEE Software*, 2(2), 9-22, March 1985.

[Lio96] J.L. Lions, "Ariane 5, Flight 501 Failure, Report by the Inquiry Board," European Space Agency, http://www.esa.it, 1996.

[LMZ02] L. Liu, H. Miao, and X. Zhan, "A Framework for Specification-Based Class Testing," *Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'02)*, 2002.

[LPU02a] B. Legeard, F. Peureux, and M. Utting, "A Comparison of the BTT and TTF Test-Generation Methods," *LNCS 2272*, pp.309-329, Springer-Verlag, 2002.

[LPU02b] B. Legeard, F. Peureux, and M. Utting, "Automated Boundary Testing from Z and B," *FME 2002, LNCS 2391*, pp.21-40, Springer-Verlag, 2002.

[LS00] B. Littlewood, L. Strigini, "Software Reliability and Dependability: a Roadmap," *Proceedings of the Conference on the Future of Software Engineering*, ACM Press, 2000.

[LT93] N.G. Leveson, C.S. Turner, "An Investigation of the Therac-25 Accidents," *IEEE Computer*, Vol. 26, No. 7, pp. 18-41, 1993.

[Meu98] C. Meudec, *Automatic Generation of Software Test Cases From Formal Specifications,* Ph.D. dissertation, The Queen's University of Belfast, May 1998.

[Mik95] E. Mikk, "Compilation of Z Specifications into C for Automatic Test Result Evaluation," *Proceedings of the 9th International Conference of Z Users,* 1995, Springer-Verlag.

[ML06] H. Miao, L. Liu, "An Approach to Formalizing Specification-Based Class Testing," *Journal of Shanghai University (English Edition),* Vol. 10, No. 1, Feb. 2006.

[MMA06] N. Mahmood, Y. Mahmood, A. Aslam, *SpecTGS Implementation,* B.S. Project, Mohammad Ali Jinnah University, Islamabad, Pakistan, Nov. 2006.

[Mye79] G.J. Myers, *The Art of Software Testing,* John Wiley & Sons, 1979, ISBN 0-471-04328-1.

[NL06] A. Nadeem, M.R. Lyu, "A Framework for Inheritance Testing from VDM++ Specifications," to appear in proceedings of the *12th IEEE International Symposium on Pacific Rim Dependable Computing (PRDC 2006),* Dec., 2006, Riverside, California, USA.

[NML06] Aamer Nadeem, Zafar I. Malik, Michael R. Lyu, "An Automated Approach to Inheritance and Polymorphic Testing using a VDM++ Specification," proceedings of the *Tenth IEEE International Multi-topic Conference (INMIC'06),* December 23-24, 2006, Islamabad, Pakistan.

[NR04] A. Nadeem, M.J. Rehman, "A Framework for Automated Testing from VDM-SL Specifications," in proceedings of the *8th IEEE International Multi-topic Conference (INMIC 2004),* Dec. 24-26, 2004, Lahore, Pakistan.

[NR05] A. Nadeem, M.J. Rehman, "TESTAF: A Test Automation Framework for Class Testing using Object-oriented Formal Specifications," *Journal of Universal Computer Science (J.UCS),* Vol. 11, No. 6, Springer-Verlag, 2005.

[OAWXH01] J. Offutt, R. Alexander, Y. Wu, Q. Xiao, C. Hutchinson, "A Fault Model for Subtype Inheritance and Polymorphism," *The Twelfth IEEE Symposium on Software Reliability Engineering, ISSRE '01,* pages 84-95, Hong Kong, P.R.C., November 2001.

[OB88] T.J. Ostrand, M.J. Balcer, "The category-partition method for specifying and generating functional tests," *Communications of the ACM,* 31(6), pp. 676-686, June 1988.

[Off98] A.J. Offutt, "Software Testing: From Theory to Practice," *IEEE AES Systems Magazine,* March 1998.

[OL99] A. J. Offutt, S. Liu, "Generating Test Data from SOFL Specifications," *The Journal of Systems and Software,* 1999, pp. 49-62.

[OLAA03] J. Offutt, S. Liu, A. Abdurazik, P. Ammann, "Generating Test Data from state-based Specifications," *Journal of Software Testing, Verification and Reliability*, 2003, pp. 25-53.

[OMG05a] OMG, *Unified Modeling Language: Superstructure Specification,* version 2.0 (formal/05-07-04). Object Management Group, Inc., August 2005.

[OMG05b] OMG, *XML Metadata Interchange (XMI), MOF 2.0/XMI Mapping Specification,* version 2.1, formal/05-09-01, Object Management Group, Inc., Sep. 2005, retrieved from http://www.omg.org/technology/documents/formal/xmi.htm.

[PAFG03] O. Pilskalns, A. Andrews, R. France, and S. Ghosh, "Rigorous Testing by Merging Structural and Behavioral UML Representations," *Sixth International Conference on the Unified Modeling Language (UML'03),* pp.234-248, 2003.

[PB00] C. Pons and G. Baum, "Formal Foundations of Object-oriented Modeling Notations," *Proceedings of the 3rd International Conference on Formal Engineering Methods (ICFEM 2000),* September, 2000.

[PKT92] N. Plat, J.V. Katwijk, and H. Toetenel, "Application and Benefits of Formal Methods in Software Development," *Software Engineering Journal,* September 1992.

[PMBF04] P. Pelliccione, H. Muccini, A. Bucchiarone, and F. Facchini, “TESTOR: Deriving Test Sequences from Model-based Specifications,” 2004.

[RM89] D.J. Richardson, O. O’Malley, “Approaches to Specification-based Testing,” *Software Engineering Notes,* 14(8), pp. 86-96, ACM SIGSOFT, *Third Symposium on Software Testing, Analysis and Verification (TAV3),* 1989.

[SC93] P. Stocks and D. Carrington, “Test Template Framework: a Specification-based Testing Case Study,” *Software Engineering Notes,* ACM SIGSOFT, 18(3), pp. 11-18, 1993.

[SC96] P. Stocks and D. Carrington, “A Framework for Specification-Based Testing,” *IEEE Transactions on Software Engineering,* vol. 22, no. 11, pp. 777-793, Nov. 1996.

[SCS97] H. Singh, M. Conrad, S. Sadeghipour, “Test Case Design based on Z and the Classification Tree Method,” *Proceedings of the First International Conference on Formal Engineering Methods,* Hiroshima, Japan, IEEE Computer Society, 1997.

[Sto93] Philip A. Stocks, *Applying Formal Methods to Software Testing*, PhD dissertation, Department of Computer Science, University of Queensland, December 1993.

[TR93] C.D. Turner, D.J. Robson, “A Suite of Tools for the State-based Testing of Object-oriented Programs”, *TR-14/92, Technical Report*, Computer Science Division, School of Engineering and Computer Science (SECS), University of Durham, Durham, England, April 1993.

[TVK90] W.T. Tsai, D. Volovik, T.F. Keefe, “Automated Test Case Generation for Programs specified by Relational Algebra Queries,” *IEEE Transactions on Software Engineering*, March 1990.

[Vag96] T. Vagoun, “Input Domain Partitioning in Software Testing,” *Proceedings of the 29th Annual Hawaii International Conference on System Sciences,* IEEE, 1996.

[WCO03] Y. Wu, M. Chen, J. Offutt, “UML-based Integration Testing for Component-based Software,” *Proceedings of the 2nd International*

Conference on COTS-Based Software Systems (ICCBSS), Ottawa, Canada, Feb. 2003.

[WGS94] E. Weyuker, T. Goradia, A. Singh, "Automatically Generating Test Data from a Boolean Specification," *IEEE Transactions on Software Engineering*, May 1994.

[WM01] J. Wittevrongel, F. Maurer, "Using UML to Partially Automate Generation of Scenario-Based Test Drivers," *Proceedings of the Object-Oriented Information Systems (OOIS'01)*, 2001.

Appendix-I

Test Cases for the *add* method of the *NNComplex* Class

```
BEGIN TEST add.1
 set_re <0>
 set_im <0>
 add <0>
END TEST

BEGIN TEST add.2
 set_re <0>
 set_im <0>
 add <1>
END TEST

BEGIN TEST add.3
 set_re <0>
 set_im <0>
 add <9>
END TEST

BEGIN TEST add.4
 set_re <0>
 set_im <1>
 add <0>
END TEST

BEGIN TEST add.5
 set_re <0>
 set_im <1>
 add <1>
END TEST
```

```
BEGIN TEST add.6
 set_re <0>
 set_im <1>
 add <9>
END TEST

BEGIN TEST add.7
 set_re <0>
 set_im <8>
 add <0>
END TEST

BEGIN TEST add.8
 set_re <0>
 set_im <8>
 add <1>
END TEST

BEGIN TEST add.9
 set_re <0>
 set_im <8>
 add <9>
END TEST

BEGIN TEST add.10
 set_re <1>
 set_im <0>
 add <-1>
END TEST

BEGIN TEST add.11
 set_re <1>
 set_im <1>
 add <0>
END TEST

BEGIN TEST add.12
 set_re <1>
 set_im <8>
 add <12>
END TEST

BEGIN TEST add.13
 set_re <5>
 set_im <0>
 add <-5>
END TEST
```

```
BEGIN TEST add.14
 set_re <5>
 set_im <1>
 add <-4>
END TEST

BEGIN TEST add.15
 set_re <5>
 set_im <8>
 add <6>
END TEST
```

Appendix-II

VDM++ Specification of *CSLaM* Case Study

```
class CSL

  instance variables
    cabDisplay    : CabDisplay;
    emergencyBrake: EmergencyBrake;
    onboardComp   : OnBoardComp;
    announcements    : seq of AnnounceBeacon;
    speedRestrictions: seq of LimitBeacon;
    firstSpeedRestriction: bool;

  inv
    len speedRestrictions <= 5;

  values
    maxSpeed: real = 180;

  operations
    public HeadMeetsBeacon(b: Beacon)
    pre  isofclass(LimitBeacon, b) => len announcements > 0;
    post (isofclass(AnnounceBeacon, b) =>
          AnnounceSpeedRestriction(b))
         and (isofclass(LimitBeacon, b) => AddSpeedRestriction(b))
         and (isofclass(CancelBeacon, b)=> DeleteAnnouncements());

    public TailMeetsBeacon(b: Beacon)
    pre ((isofclass(LimitBeacon, b) and not firstSpeedRestriction)
        or isofclass(EndBeacon, b)) => len speedRestrictions > 0;
    post (isofclass(LimitBeacon, b) => TailMeetsLimitBeacon(b))
       and (isofclass(EndBeacon, b) => TailMeetsEndBeacon());

    public AnnounceSpeedRestriction(b: AnnounceBeacon)
    ext wr cabDisplay: CabDisplay;
           announcements: seq of AnnounceBeacon;
    post announcements = announcements' ^ [b] and
         not cabDisplay.GetDisplay()(3);
```

```
public AddSpeedRestriction(b: LimitBeacon)
ext wr announcements: seq of AnnounceBeacon;
       speedRestrictions: seq of LimitBeacon;
       cabDisplay: CabDisplay;
pre len announcements > 0;
post len speedRestrictions’ < 5 =>
     (b.GetSpeedRestriction = (hd announcements).GetTargetSpeed()
     and (speedRestrictions = speedRestrictions’ ^ [b])
     and (announcements = tl announcements’)
     and not cabDisplay.GetDisplay()(3))
     and not(len speedRestrictions’ < 5) =>
             cabDisplay.GetDisplay()(3);

public DeleteAnnouncements()
ext wr announcements: seq of AnnounceBeacon;
       cabDisplay: CabDisplay;
post announcements = [] and not cabDisplay.GetDisplay()(3);

public TailMeetsLimitBeacon(b: LimitBeacon)
ext wr firstSpeedRestriction: bool;
       speedRestrictions: seq of LimitBeacon;
       cabDisplay: CabDisplay;
pre not firstSpeedRestriction) => len speedRestrictions > 0;
post (not firstSpeedRestriction’ =>
     speedRestrictions = tl speedRestrictions’
     and not cabDisplay.GetDisplay()(3))
     and (firstSpeedRestriction’ => not firstSpeedRestriction);

public TailMeetsEndBeacon()
ext wr firstSpeedRestriction: bool;
       speedRestrictions: seq of LimitBeacon;
       cabDisplay: CabDisplay;
pre len speedRestrictions > 0;
post firstSpeedRestriction
     and speedRestrictions = tl speedRestrictions’
     and not cabDisplay.GetDisplay()(3);

public NoBeaconMet()
ext wr announcements: seq of AnnounceBeacon;
       cabDisplay: CabDisplay;
pre len announcements > 0;
post announcements = tl announcements’
     and cabDisplay.GetDisplay()(3);

public DeletePossibleGroundFault()
ext wr cabDisplay: CabDisplay;
post not cabDisplay.GetDisplay()(3);

public CheckSpeed(speed)
ext rd onboardComputer: OnboardComputer;
    wr emergencyBrake: EmergencyBrake;
    wr cabDisplay: CabDisplay;
post let speedAlarm = onboardComp.CheckSpeed(speed, GetMaxSpeed())
     in ((speedAlarm = <speedOk> and
     not emergencyBrake.GetEmergencyBrake()) =>
         not cabDisplay.GetDisplay()(1))
     and ((speedAlarm = <AlarmSpeed> and
```

```
          not emergencyBrake.GetEmergencyBrake()) =>
              cabDisplay.GetDisplay()(1))
          and (speedAlarm = <EmergencyBrakeSpeed> =>
            (cabDisplay.GetDisplay()(2) and
             emergencyBrake.GetEmergencyBrake());

  public GetMaxSpeed() mSpeed: real
  post
    (len speedRestrictions > 0 =>
          let speeds = { limit.GetSpeedRestriction()
                       | limit in set elems speedRestrictions } in
          let minspeed in set speeds be st forall sp in set speeds &
              minspeed <= sp in
          mSpeed = minspeed) and
    (len speedRestrictions = 0 => mSpeed = maxSpeed;

  public ReleaseEmergencyBrake(sp: real)
  ext wr cabDisplay: CabDisplay;
         emergencyBrake: EmergencyBrake;
  pre cabDisplay.GetGetDisplay()(2) and
    emergencyBrake.GetEmergencyBrake()
  post (sp=0) => (not cabDisplay.GetDisplay()(2)
       and not emergencyBrake.GetEmergencyBrake());

  public GetCabDisplay() cabDisp: CabDisplay
  ext rd cabDisplay: CabDisplay;
  post cabDisp = cabDisplay;

  public GetEmergencyBrake() emBrake: EmergencyBrake
  ext rd emergencyBrake: EmergencyBrake;
  post emBrake = emergencyBrake;

  public GetAnnouncements() ann: seq of AnnounceBeacon
  ext rd announcements: seq of AnnounceBeacon;
  post ann = announcements;

  public GetSpeedRestrictions() rest: seq of LimitBeacon
  ext rd speedRestrictions: seq of LimitBeacon;
  post rest = speedRestrictions;

end CSL

class OnBoardComp

  types
    public AlarmLevel = <SpeedOk> | <AlarmSpeed> |
<EmergencyBrakeSpeed>;

  values
    alarmSpeedAdd = 5;
    emergencySpeedAdd = 10;

  operations
    public CheckSpeed(speed: real, maxSpeed: real) AL: AlarmLevel
    post ((speed<maxSpeed+alarmSpeedAdd)=>(AL=<SpeedOk>) and
         ((speed>=maxSpeed+alarmSpeedAdd) and
         (speed<maxSpeed+emergencySpeedAdd))=>(AL=<AlarmSpeed>) and
```

```
        (speed>=maxSpeed+emergencySpeedAdd)=>(AL=<EmergencyBrakeSpeed>))

end OnBoardComp

class CabDisplay

  instance variables
    alarm: bool;
    emergencyBrake: bool;
    groundFault: bool;

  operations
    public SetAlarm()
    ext wr alarm: bool
    post alarm;

    public UnsetAlarm()
    ext wr alarm: bool
    post not alarm;

    public SetEmergencyBrake()
    ext wr emergencyBrake: bool
    post emergencyBrake;

    public UnsetEmergencyBrake()
    ext wr emergencyBrake: bool
    post not emergencyBrake;

    public SetGroundFault()
    ext wr groundFault: bool
    post groundFault;

    public UnsetGroundFault()
    ext wr groundFault: bool
    post not groundFault;

    public GetDisplay() disp: seq of bool
    ext rd alarm: bool
           emergencyBrake: bool;
           groundFault: bool;
    post disp=mk_(alarm,emergencyBrake,groundFault);

end CabDisplay

class EmergencyBrake

  instance variables
    emergencyBrake: bool;

  operations
    public SetEmergencyBrake: ()
    ext wr emergencyBrake: bool;
    post emergencyBrake;

    public UnsetEmergencyBrake: () ==> ()
    ext wr emergencyBrake: bool;
    post not emergencyBrake;
```

```
    public GetEmergencyBrake: () EB: bool;
    ext rd emergencyBrake: bool;
    post EB = emergencyBrake;

end EmergencyBrake

class Beacon

end Beacon

class AnnounceBeacon is subclass of Beacon

  instance variables
    targetspeed: real;

  operations
    public AnnounceBeacon: real ==> AnnounceBeacon
    public AnnounceBeacon(ts: real) AB: AnnounceBeacon
    ext rd self
        wr targetspeed: real
    post (targetspeed = ts) and (AB = self)

public GetTargetSpeed() TS: real
ext rd targetspeed
      post TS = targetspeed
end AnnounceBeacon

class LimitBeacon is subclass of Beacon

  instance variables
    speed: real;

  operations
    public SetSpeedRestriction(s: real)
    ext wr speed: real
    post speed = s

    public GetSpeedRestriction() SR: real
    ext rd speed: real
    post SR = speed

end LimitBeacon

class CancelBeacon is subclass of Beacon

end CancelBeacon

class EndBeacon is subclass of Beacon

end EndBeacon
```

Appendix-III

XMI Output for Communication Diagram of CSLaM Case Study

This appendix lists the XMI output generated by the Borland's Together tool for the *Communication diagram* of CSLaM case study in chapter 7.

```
<?xml version = '1.0' encoding = 'ASCII' ?>
<XMI xmi.version = '1.1' xmlns:UML = '//org.omg/UML/1.3'>
 <XMI.header>
  <XMI.documentation>
   <XMI.exporter>
       TogetherSoft
   </XMI.exporter>
   <XMI.exporterVersion>
       6.0
   </XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name = 'UML' xmi.version = '1.4'/>
 </XMI.header>
 <XMI.content>
  <UML:Model xmi.id = 'S.1' name = 'Project' visibility = 'public'>
   <UML:Namespace.ownedElement>
    <!--From Class EmergencyBrake to Class CSL-->
    <UML:Association xmi.id = 'G.6'
          name = '{EmergencyBrake-CSL}' visibility = 'private' isSpecification
= 'false'
          isAbstract = 'false'>
     <UML:Association.connection>
      <UML:AssociationEnd xmi.id = 'G.10' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.9'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = 'G.11' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
```

```
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.10'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
     </UML:Association.connection>
    </UML:Association>
    <!--From Class CSL to Class OnboardComputer-->
    <UML:Association xmi.id = 'G.7'
          name = '{CSL-OnboardComputer}' visibility = 'private' isSpecification
= 'false'
          isAbstract = 'false'>
     <UML:Association.connection>
      <UML:AssociationEnd xmi.id = 'G.12' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.10'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = 'G.13' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.12'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
     </UML:Association.connection>
    </UML:Association>
    <!--From Class CabDisplay to Class CSL-->
    <UML:Association xmi.id = 'G.8'
          name = '{CabDisplay-CSL}' visibility = 'private' isSpecification =
'false'
          isAbstract = 'false'>
     <UML:Association.connection>
      <UML:AssociationEnd xmi.id = 'G.14' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.11'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = 'G.15' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
```

```
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.10'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
     </UML:Association.connection>
    </UML:Association>
    <!--From Class Beacon to Class CSL-->
    <UML:Association xmi.id = 'G.9'
          name = '{Beacon-CSL}' visibility = 'private' isSpecification =
'false'
          isAbstract = 'false'>
     <UML:Association.connection>
      <UML:AssociationEnd xmi.id = 'G.16' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.13'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = 'G.17' visibility = 'public' isSpecification
= 'false'
            isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
            targetScope = 'instance' changeability = 'changeable'>
       <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity/>
       </UML:AssociationEnd.multiplicity>
       <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref = 'S.10'/>
       </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
     </UML:Association.connection>
    </UML:Association>
    <UML:Class xmi.id = 'S.14'
          name = 'AnnounceBeacon' visibility = 'public' isSpecification =
'false'
          isAbstract = 'false' isActive = 'false'>
     <UML:Classifier.feature>
      <UML:Operation xmi.id = 'S.18'
            name = 'AnnounceBeacon' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.26' name = 'AnnounceBeacon.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.19'
            name = 'GetTargetSpeed' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.27' name = 'GetTargetSpeed.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
     </UML:Classifier.feature>
```

```
 <UML:Namespace.ownedElement>
  <UML:Generalization xmi.id = 'G.19'
        name = '' visibility = 'public' isSpecification = 'false'
        discriminator = ''>
   <UML:Generalization.child>
    <UML:GeneralizableElement xmi.idref = 'S.14'/>
   </UML:Generalization.child>
   <UML:Generalization.parent>
    <UML:GeneralizableElement xmi.idref = 'S.13'/>
   </UML:Generalization.parent>
  </UML:Generalization>
 </UML:Namespace.ownedElement>
</UML:Class>
<UML:Class xmi.id = 'S.13'
      name = 'Beacon' visibility = 'public' isSpecification = 'false'
      isAbstract = 'false' isActive = 'false'>
 <UML:Classifier.feature>
  <UML:Attribute xmi.id = 'S.20'
        name = 'lnkCSL' visibility = 'private' isSpecification = 'false'
        changeability = 'changeable' ownerScope = 'instance'>
   <UML:StructuralFeature.multiplicity>
    <UML:Multiplicity>
     <UML:Multiplicity.range>
      <UML:MultiplicityRange lower = '1' upper = '1'/>
     </UML:Multiplicity.range>
    </UML:Multiplicity>
   </UML:StructuralFeature.multiplicity>
   <UML:StructuralFeature.type>
    <UML:Classifier>
     <UML:Namespace.ownedElement>
      <UML:DataType xmi.idref = 'G.20'/>
     </UML:Namespace.ownedElement>
    </UML:Classifier>
   </UML:StructuralFeature.type>
  </UML:Attribute>
 </UML:Classifier.feature>
</UML:Class>
<UML:Class xmi.id = 'S.11'
      name = 'CabDisplay' visibility = 'public' isSpecification = 'false'
      isAbstract = 'false' isActive = 'false'>
 <UML:Classifier.feature>
  <UML:Attribute xmi.id = 'S.21'
        name = 'lnkCSL' visibility = 'private' isSpecification = 'false'
        changeability = 'changeable' ownerScope = 'instance'>
   <UML:StructuralFeature.multiplicity>
    <UML:Multiplicity>
     <UML:Multiplicity.range>
      <UML:MultiplicityRange lower = '1' upper = '1'/>
     </UML:Multiplicity.range>
    </UML:Multiplicity>
   </UML:StructuralFeature.multiplicity>
   <UML:StructuralFeature.type>
    <UML:Classifier>
     <UML:Namespace.ownedElement>
      <UML:DataType xmi.idref = 'G.20'/>
     </UML:Namespace.ownedElement>
    </UML:Classifier>
   </UML:StructuralFeature.type>
  </UML:Attribute>
  <UML:Operation xmi.id = 'S.22'
        name = 'SetAlarm' visibility = 'public'
        isSpecification = 'false'
        isAbstract = 'false' ownerScope = 'instance'>
```

```
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.28' name = 'SetAlarm.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.23'
            name = 'UnsetAlarm' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.29' name = 'UnsetAlarm.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.24'
            name = 'SetEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.30' name = 'SetEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.25'
            name = 'UnsetEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.31' name = 'UnsetEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.26'
            name = 'SetGroundFault' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.32' name = 'SetGroundFault.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.27'
            name = 'UnsetGroundFault' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.33' name = 'UnsetGroundFault.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.28'
            name = 'GetDisplay' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.34' name = 'GetDisplay.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
```

```
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
     </UML:Classifier.feature>
    </UML:Class>
    <UML:Class xmi.id = 'S.16'
          name = 'CancelBeacon' visibility = 'public' isSpecification = 'false'
          isAbstract = 'false' isActive = 'false'>
     <UML:Namespace.ownedElement>
      <UML:Generalization xmi.id = 'G.21'
            name = '' visibility = 'public' isSpecification = 'false'
            discriminator = ''>
       <UML:Generalization.child>
        <UML:GeneralizableElement xmi.idref = 'S.16'/>
       </UML:Generalization.child>
       <UML:Generalization.parent>
        <UML:GeneralizableElement xmi.idref = 'S.13'/>
       </UML:Generalization.parent>
      </UML:Generalization>
     </UML:Namespace.ownedElement>
    </UML:Class>
    <UML:Class xmi.id = 'S.10'
          name = 'CSL' visibility = 'public' isSpecification = 'false'
          isAbstract = 'false' isActive = 'false'>
     <UML:Classifier.feature>
      <UML:Operation xmi.id = 'S.29'
            name = 'HeadMeetsBeacon' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.35' name = 'HeadMeetsBeacon.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.30'
            name = 'TailMeetsBeacon' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.36' name = 'TailMeetsBeacon.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.31'
            name = 'AnnounceSpeedRestriction' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.37' name =
'AnnounceSpeedRestriction.Return' isSpecification = 'false' kind = 'return'
type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.32'
            name = 'AddSpeedRestriction' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.38' name = 'AddSpeedRestriction.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
```

```
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.33'
            name = 'DeleteAnnouncements' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.39' name = 'DeleteAnnouncements.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.34'
            name = 'TailMeetsLimitBeacon' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.40' name = 'TailMeetsLimitBeacon.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.35'
            name = 'TailMeetsEndBeacon' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.41' name = 'TailMeetsEndBeacon.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.36'
            name = 'NoBeaconMet' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.42' name = 'NoBeaconMet.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.37'
            name = 'DeletePossibleGroundFault' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.43' name =
'DeletePossibleGroundFault.Return' isSpecification = 'false' kind = 'return'
type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.38'
            name = 'CheckSpeed' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.44' name = 'CheckSpeed.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
```

```
      </UML:Operation>
      <UML:Operation xmi.id = 'S.39'
            name = 'GetMaxSpeed' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.45' name = 'GetMaxSpeed.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.40'
            name = 'ReleaseEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.46' name = 'ReleaseEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.41'
            name = 'RaiseGroungFault' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.47' name = 'RaiseGroungFault.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.42'
            name = 'GetCabDisplay' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.48' name = 'GetCabDisplay.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.43'
            name = 'GetEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.49' name = 'GetEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.44'
            name = 'GetAnnouncements' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
       <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = 'XX.50' name = 'GetAnnouncements.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
       </UML:BehavioralFeature.parameter>
      </UML:Operation>
      <UML:Operation xmi.id = 'S.45'
            name = 'GetSpeedRestrictions' visibility = 'public'
```

```
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.51' name = 'GetSpeedRestrictions.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
         </UML:Parameter>
        </UML:BehavioralFeature.parameter>
       </UML:Operation>
       <UML:Attribute xmi.id = 'S.46'
            name = 'lnkOnboardComputer' visibility = 'private' isSpecification
= 'false'
            changeability = 'changeable' ownerScope = 'instance'>
        <UML:StructuralFeature.multiplicity>
         <UML:Multiplicity>
          <UML:Multiplicity.range>
           <UML:MultiplicityRange lower = '1' upper = '1'/>
          </UML:Multiplicity.range>
         </UML:Multiplicity>
        </UML:StructuralFeature.multiplicity>
        <UML:StructuralFeature.type>
         <UML:Classifier>
          <UML:Namespace.ownedElement>
           <UML:DataType xmi.idref = 'G.22'/>
          </UML:Namespace.ownedElement>
         </UML:Classifier>
        </UML:StructuralFeature.type>
       </UML:Attribute>
      </UML:Classifier.feature>
     </UML:Class>
     <UML:Class xmi.id = 'S.9'
          name = 'EmergencyBrake' visibility = 'public' isSpecification =
'false'
          isAbstract = 'false' isActive = 'false'>
      <UML:Classifier.feature>
       <UML:Operation xmi.id = 'S.47'
            name = 'UnsetEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.52' name = 'UnsetEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
         </UML:Parameter>
        </UML:BehavioralFeature.parameter>
       </UML:Operation>
       <UML:Operation xmi.id = 'S.48'
            name = 'GetEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.53' name = 'GetEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
         </UML:Parameter>
        </UML:BehavioralFeature.parameter>
       </UML:Operation>
       <UML:Operation xmi.id = 'S.49'
            name = 'SetEmergencyBrake' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.54' name = 'SetEmergencyBrake.Return'
isSpecification = 'false' kind = 'return' type = 'G.23'>
         </UML:Parameter>
        </UML:BehavioralFeature.parameter>
```

```
      </UML:Operation>
      <UML:Attribute xmi.id = 'S.50'
            name = 'emergenceyBrake' visibility = 'private' isSpecification =
'false'
            changeability = 'changeable' ownerScope = 'instance'>
       <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity>
         <UML:Multiplicity.range>
          <UML:MultiplicityRange lower = '1' upper = '1'/>
         </UML:Multiplicity.range>
        </UML:Multiplicity>
       </UML:StructuralFeature.multiplicity>
       <UML:StructuralFeature.type>
        <UML:Classifier>
         <UML:Namespace.ownedElement>
          <UML:DataType xmi.idref = 'G.23'/>
         </UML:Namespace.ownedElement>
        </UML:Classifier>
       </UML:StructuralFeature.type>
      </UML:Attribute>
      <UML:Attribute xmi.id = 'S.51'
            name = 'lnkCSL' visibility = 'private' isSpecification = 'false'
            changeability = 'changeable' ownerScope = 'instance'>
       <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity>
         <UML:Multiplicity.range>
          <UML:MultiplicityRange lower = '1' upper = '1'/>
         </UML:Multiplicity.range>
        </UML:Multiplicity>
       </UML:StructuralFeature.multiplicity>
       <UML:StructuralFeature.type>
        <UML:Classifier>
         <UML:Namespace.ownedElement>
          <UML:DataType xmi.idref = 'G.20'/>
         </UML:Namespace.ownedElement>
        </UML:Classifier>
       </UML:StructuralFeature.type>
      </UML:Attribute>
     </UML:Classifier.feature>
    </UML:Class>
    <UML:Class xmi.id = 'S.17'
          name = 'EndBeacon' visibility = 'public' isSpecification = 'false'
          isAbstract = 'false' isActive = 'false'>
     <UML:Namespace.ownedElement>
      <UML:Generalization xmi.id = 'G.24'
            name = '' visibility = 'public' isSpecification = 'false'
            discriminator = ''>
       <UML:Generalization.child>
        <UML:GeneralizableElement xmi.idref = 'S.17'/>
       </UML:Generalization.child>
       <UML:Generalization.parent>
        <UML:GeneralizableElement xmi.idref = 'S.13'/>
       </UML:Generalization.parent>
      </UML:Generalization>
     </UML:Namespace.ownedElement>
    </UML:Class>
    <UML:Class xmi.id = 'S.15'
          name = 'LimitBeacon' visibility = 'public' isSpecification = 'false'
          isAbstract = 'false' isActive = 'false'>
     <UML:Classifier.feature>
      <UML:Operation xmi.id = 'S.52'
            name = 'SetSpeedRestriction' visibility = 'public'
            isSpecification = 'false'
```

```
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.55' name = 'SetSpeedRestriction.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
        </UML:BehavioralFeature.parameter>
       </UML:Operation>
       <UML:Operation xmi.id = 'S.53'
            name = 'GetSpeedRestriction' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.56' name = 'GetSpeedRestriction.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
        </UML:BehavioralFeature.parameter>
       </UML:Operation>
      </UML:Classifier.feature>
      <UML:Namespace.ownedElement>
       <UML:Generalization xmi.id = 'G.25'
            name = '' visibility = 'public' isSpecification = 'false'
            discriminator = ''>
        <UML:Generalization.child>
         <UML:GeneralizableElement xmi.idref = 'S.15'/>
        </UML:Generalization.child>
        <UML:Generalization.parent>
         <UML:GeneralizableElement xmi.idref = 'S.13'/>
        </UML:Generalization.parent>
       </UML:Generalization>
      </UML:Namespace.ownedElement>
     </UML:Class>
     <UML:Class xmi.id = 'S.12'
          name = 'OnboardComputer' visibility = 'public' isSpecification =
'false'
          isAbstract = 'false' isActive = 'false'>
      <UML:Classifier.feature>
       <UML:Operation xmi.id = 'S.54'
            name = 'CheckSpeed' visibility = 'public'
            isSpecification = 'false'
            isAbstract = 'false' ownerScope = 'instance'>
        <UML:BehavioralFeature.parameter>
         <UML:Parameter xmi.id = 'XX.57' name = 'CheckSpeed.Return'
isSpecification = 'false' kind = 'return' type = 'G.18'>
        </UML:Parameter>
        </UML:BehavioralFeature.parameter>
       </UML:Operation>
      </UML:Classifier.feature>
     </UML:Class>
     <UML:Collaboration xmi.id = 'S.3'
          name = 'Collaboration' visibility = 'public' isSpecification =
'false' isAbstract = 'false'>
      <UML:Namespace.ownedElement>
       <UML:ClassifierRole xmi.id = 'G.1'
            name = 'Object2' visibility = 'package' isSpecification = 'false'
            isAbstract = 'false'>
        <UML:ClassifierRole.multiplicity>
         <UML:Multiplicity>
          <UML:Multiplicity.range>
           <UML:MultiplicityRange lower = '1' upper = '1'/>
          </UML:Multiplicity.range>
         </UML:Multiplicity>
        </UML:ClassifierRole.multiplicity>
        <UML:ClassifierRole.base>
```

```
   <UML:Classifier xmi.idref = 'S.14'/>
  </UML:ClassifierRole.base>
 </UML:ClassifierRole>
 <UML:ClassifierRole xmi.id = 'G.2'
       name = 'Object3' visibility = 'package' isSpecification = 'false'
       isAbstract = 'false'>
  <UML:ClassifierRole.multiplicity>
   <UML:Multiplicity>
    <UML:Multiplicity.range>
     <UML:MultiplicityRange lower = '1' upper = '1'/>
    </UML:Multiplicity.range>
   </UML:Multiplicity>
  </UML:ClassifierRole.multiplicity>
  <UML:ClassifierRole.base>
   <UML:Classifier xmi.idref = 'S.11'/>
  </UML:ClassifierRole.base>
 </UML:ClassifierRole>
 <UML:ClassifierRole xmi.id = 'G.3'
       name = 'Object4' visibility = 'package' isSpecification = 'false'
       isAbstract = 'false'>
  <UML:ClassifierRole.multiplicity>
   <UML:Multiplicity>
    <UML:Multiplicity.range>
     <UML:MultiplicityRange lower = '1' upper = '1'/>
    </UML:Multiplicity.range>
   </UML:Multiplicity>
  </UML:ClassifierRole.multiplicity>
  <UML:ClassifierRole.base>
   <UML:Classifier xmi.idref = 'S.15'/>
  </UML:ClassifierRole.base>
 </UML:ClassifierRole>
 <UML:ClassifierRole xmi.id = 'G.4'
       name = 'Object5' visibility = 'package' isSpecification = 'false'
       isAbstract = 'false'>
  <UML:ClassifierRole.multiplicity>
   <UML:Multiplicity>
    <UML:Multiplicity.range>
     <UML:MultiplicityRange lower = '1' upper = '1'/>
    </UML:Multiplicity.range>
   </UML:Multiplicity>
  </UML:ClassifierRole.multiplicity>
  <UML:ClassifierRole.base>
   <UML:Classifier/>
  </UML:ClassifierRole.base>
 </UML:ClassifierRole>
 <UML:ClassifierRole xmi.id = 'G.5'
       name = 'Object1' visibility = 'package' isSpecification = 'false'
       isAbstract = 'false'>
  <UML:ClassifierRole.multiplicity>
   <UML:Multiplicity>
    <UML:Multiplicity.range>
     <UML:MultiplicityRange lower = '1' upper = '1'/>
    </UML:Multiplicity.range>
   </UML:Multiplicity>
  </UML:ClassifierRole.multiplicity>
  <UML:ClassifierRole.base>
   <UML:Classifier xmi.idref = 'S.10'/>
  </UML:ClassifierRole.base>
 </UML:ClassifierRole>
</UML:Namespace.ownedElement>
<UML:Collaboration.interaction>
 <UML:Interaction xmi.id = 'G.0'
```

```
            name = '{Logical View}Collaboration' visibility = 'public'
isSpecification = 'false'>
       <UML:Interaction.message>
        <UML:Message xmi.id = 'G.26'
              name = 'HeadMeetsBeacon' visibility = 'package' isSpecification =
'false' sender = 'G.4' receiver = 'G.5'>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.251'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.27'
              name = 'AnnounceSpeedRestriction' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.5'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.26'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.253'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.30'
              name = 'DeletePossibleGroundFault' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.5'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.27'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.255'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.31'
              name = 'GetDisplay' visibility = 'package' isSpecification =
'false' sender = 'G.5' receiver = 'G.2'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.30'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.257'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.32'
              name = 'UnsetGroundFault' visibility = 'package' isSpecification
= 'false' sender = 'G.5' receiver = 'G.2'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.31'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.259'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.28'
              name = 'AddSpeedRestriction' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.5'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.32'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.261'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.33'
              name = 'GetTargetSpeed' visibility = 'package' isSpecification =
'false' sender = 'G.5' receiver = 'G.1'>
```

```
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.28'/>
          </UML:Message.predecessor>
          <UML:Message.action>
           <UML:Action xmi.idref = 'XX.263'/>
          </UML:Message.action>
         </UML:Message>
         <UML:Message xmi.id = 'G.36'
              name = 'SetSpeedRestriction' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.3'>
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.33'/>
          </UML:Message.predecessor>
          <UML:Message.action>
           <UML:Action xmi.idref = 'XX.265'/>
          </UML:Message.action>
         </UML:Message>
         <UML:Message xmi.id = 'G.37'
              name = 'DeletePossibleGroundFault' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.5'>
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.36'/>
          </UML:Message.predecessor>
          <UML:Message.action>
           <UML:Action xmi.idref = 'XX.267'/>
          </UML:Message.action>
         </UML:Message>
         <UML:Message xmi.id = 'G.38'
              name = 'GetDisplay' visibility = 'package' isSpecification =
'false' sender = 'G.5' receiver = 'G.2'>
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.37'/>
          </UML:Message.predecessor>
          <UML:Message.action>
           <UML:Action xmi.idref = 'XX.269'/>
          </UML:Message.action>
         </UML:Message>
         <UML:Message xmi.id = 'G.39'
              name = 'UnsetGroundFault' visibility = 'package' isSpecification
= 'false' sender = 'G.5' receiver = 'G.2'>
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.38'/>
          </UML:Message.predecessor>
          <UML:Message.action>
           <UML:Action xmi.idref = 'XX.271'/>
          </UML:Message.action>
         </UML:Message>
         <UML:Message xmi.id = 'G.34'
              name = 'RaiseGroungFault' visibility = 'package' isSpecification
= 'false' sender = 'G.5' receiver = 'G.5'>
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.39'/>
          </UML:Message.predecessor>
          <UML:Message.action>
           <UML:Action xmi.idref = 'XX.273'/>
          </UML:Message.action>
         </UML:Message>
         <UML:Message xmi.id = 'G.35'
              name = 'SetGroundFault' visibility = 'package' isSpecification =
'false' sender = 'G.5' receiver = 'G.2'>
          <UML:Message.predecessor>
           <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.34'/>
          </UML:Message.predecessor>
```

```
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.275'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.29'
              name = 'DeleteAnnouncements' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.5'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.35'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.277'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.40'
              name = 'DeletePossibleGroundFault' visibility = 'package'
isSpecification = 'false' sender = 'G.5' receiver = 'G.5'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.29'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.279'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.41'
              name = 'GetDisplay' visibility = 'package' isSpecification =
'false' sender = 'G.5' receiver = 'G.2'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.40'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.281'/>
         </UML:Message.action>
        </UML:Message>
        <UML:Message xmi.id = 'G.42'
              name = 'UnsetGroundFault' visibility = 'package' isSpecification
= 'false' sender = 'G.5' receiver = 'G.2'>
         <UML:Message.predecessor>
          <Behavioral_Elements.Collaborations.Message xmi.idref = 'G.41'/>
         </UML:Message.predecessor>
         <UML:Message.action>
          <UML:Action xmi.idref = 'XX.283'/>
         </UML:Message.action>
        </UML:Message>
       </UML:Interaction.message>
      </UML:Interaction>
     </UML:Collaboration.interaction>
    </UML:Collaboration>
    <UML:CallAction xmi.id = 'XX.251'
          name = 'HeadMeetsBeacon' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
```

```
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.29'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.253'
          name = 'AnnounceSpeedRestriction' visibility = 'public'
isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.31'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.255'
          name = 'DeletePossibleGroundFault' visibility = 'public'
isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.37'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.257'
          name = 'GetDisplay' visibility = 'public' isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.28'/>
     </UML:CallAction.operation>
```

```
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.259'
          name = 'UnsetGroundFault' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.27'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.261'
          name = 'AddSpeedRestriction' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.32'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.263'
          name = 'GetTargetSpeed' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.19'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.265'
```

```
          name = 'SetSpeedRestriction' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.52'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.267'
          name = 'DeletePossibleGroundFault' visibility = 'public'
isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.37'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.269'
          name = 'GetDisplay' visibility = 'public' isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.28'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.271'
          name = 'UnsetGroundFault' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
```

```
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.27'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.273'
          name = 'RaiseGroungFault' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.41'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.275'
          name = 'SetGroundFault' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.26'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.277'
          name = 'DeleteAnnouncements' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
```

```
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.33'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.279'
          name = 'DeletePossibleGroundFault' visibility = 'public'
isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.37'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.281'
          name = 'GetDisplay' visibility = 'public' isSpecification = 'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.28'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:CallAction xmi.id = 'XX.283'
          name = 'UnsetGroundFault' visibility = 'public' isSpecification =
'false'
          isAsynchronous = 'false'>
     <UML:Action.recurrence>
      <UML:IterationExpression
            language = '' body = ''/>
     </UML:Action.recurrence>
     <UML:Action.target>
      <UML:ObjectSetExpression
            language = '' body = ''/>
     </UML:Action.target>
```

```
     <UML:Action.script>
      <UML:ActionExpression
            language = '' body = ''/>
     </UML:Action.script>
     <UML:CallAction.operation>
      <UML:Operation xmi.idref = 'S.27'/>
     </UML:CallAction.operation>
    </UML:CallAction>
    <UML:DataType xmi.id = 'G.18'
          name = 'void' visibility = 'public' isSpecification = 'false'/>
    <UML:DataType xmi.id = 'G.20'
          name = 'CSL' visibility = 'public' isSpecification = 'false'/>
    <UML:DataType xmi.id = 'G.22'
          name = 'OnboardComputer' visibility = 'public' isSpecification =
'false'/>
    <UML:DataType xmi.id = 'G.23'
          name = 'bool' visibility = 'public' isSpecification = 'false'/>
    <!--==================== actor [Stereotype] ====================-->
    <UML:Stereotype xmi.id = 'XX.62'
          name = 'actor' visibility = 'public' isSpecification = 'false' icon =
''>
     <UML:Stereotype.baseClass>
       ClassifierRole
     </UML:Stereotype.baseClass>
    </UML:Stereotype>
   </UML:Namespace.ownedElement>
  </UML:Model>
  <UML:TaggedValue xmi.id = 'XX.0'
        name = 'name'
        modelElement = 'G.1'>
   <UML:TaggedValue.dataValue>
       Object2
   </UML:TaggedValue.dataValue>
  </UML:TaggedValue>
  <UML:TaggedValue xmi.id = 'XX.1'
        name = 'name'
        modelElement = 'G.2'>
   <UML:TaggedValue.dataValue>
       Object3
   </UML:TaggedValue.dataValue>
  </UML:TaggedValue>
  <UML:TaggedValue xmi.id = 'XX.2'
        name = 'name'
        modelElement = 'G.3'>
   <UML:TaggedValue.dataValue>
       Object4
   </UML:TaggedValue.dataValue>
  </UML:TaggedValue>
  <UML:TaggedValue xmi.id = 'XX.3'
        name = 'name'
        modelElement = 'G.4'>
   <UML:TaggedValue.dataValue>
       Object5
   </UML:TaggedValue.dataValue>
  </UML:TaggedValue>
  <UML:TaggedValue xmi.id = 'XX.4'
        name = 'name'
        modelElement = 'G.5'>
   <UML:TaggedValue.dataValue>
       Object1
   </UML:TaggedValue.dataValue>
  </UML:TaggedValue>
  <UML:TaggedValue xmi.id = 'XX.64'
```

```
      name = 'sendingInstant'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    115
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.65'
      name = 'processingDuration'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    719
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.66'
      name = 'sequenceNumber'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.67'
      name = 'operation'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#HeadMeetsBeacon#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.68'
      name = 'operationNameAsText'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    &apos;HeadMeetsBeacon():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.69'
      name = 'minProcessingDuration'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    180
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.70'
      name = 'sendingInstant'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    124
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.71'
      name = 'processingDuration'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    151
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.72'
      name = 'sequenceNumber'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.73'
```

```
      name = 'operation'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#AnnounceSpeedRestriction#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.74'
      name = 'operationNameAsText'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    &apos;AnnounceSpeedRestriction():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.75'
      name = 'minProcessingDuration'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    30
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.76'
      name = 'sendingInstant'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    160
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.77'
      name = 'processingDuration'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    105
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.78'
      name = 'sequenceNumber'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    1.1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.79'
      name = 'minProcessingDuration'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    42
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.80'
      name = 'operation'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeletePossibleGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.81'
      name = 'operationNameAsText'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    &apos;DeletePossibleGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.82'
```

```
      name = 'sendingInstant'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    210
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.83'
      name = 'processingDuration'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.84'
      name = 'sequenceNumber'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    1.1.1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.85'
      name = 'operation'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#GetDisplay#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.86'
      name = 'operationNameAsText'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    &apos;GetDisplay():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.87'
      name = 'sendingInstant'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    235
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.88'
      name = 'processingDuration'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.89'
      name = 'sequenceNumber'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    1.1.1.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.90'
      name = 'operation'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#UnsetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.91'
```

```
      name = 'operationNameAsText'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    &apos;UnsetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.92'
      name = 'sendingInstant'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    306
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.93'
      name = 'processingDuration'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    334
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.94'
      name = 'sequenceNumber'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    1.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.95'
      name = 'operation'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#AddSpeedRestriction#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.96'
      name = 'operationNameAsText'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    &apos;AddSpeedRestriction():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.97'
      name = 'minProcessingDuration'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    114
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.98'
      name = 'sendingInstant'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    350
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.99'
      name = 'processingDuration'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.100'
```

```
      name = 'sequenceNumber'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    1.2.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.101'
      name = 'operation'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#AnnounceBeacon#GetTargetSpeed#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.102'
      name = 'operationNameAsText'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    &apos;GetTargetSpeed():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.103'
      name = 'sendingInstant'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    390
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.104'
      name = 'processingDuration'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.105'
      name = 'sequenceNumber'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    1.2.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.106'
      name = 'operation'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#LimitBeacon#SetSpeedRestriction#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.107'
      name = 'operationNameAsText'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    &apos;SetSpeedRestriction():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.108'
      name = 'sendingInstant'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    415
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.109'
```

```
      name = 'processingDuration'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    100
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.110'
      name = 'sequenceNumber'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    1.2.3
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.111'
      name = 'operation'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeletePossibleGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.112'
      name = 'operationNameAsText'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    &apos;DeletePossibleGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.113'
      name = 'minProcessingDuration'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    35
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.114'
      name = 'sendingInstant'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    460
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.115'
      name = 'processingDuration'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.116'
      name = 'sequenceNumber'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    1.2.3.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.117'
      name = 'operation'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#GetDisplay#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.118'
```

```
      name = 'operationNameAsText'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    &apos;GetDisplay():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.119'
      name = 'sendingInstant'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    485
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.120'
      name = 'processingDuration'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.121'
      name = 'sequenceNumber'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    1.2.3.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.122'
      name = 'operation'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#UnsetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.123'
      name = 'operationNameAsText'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    &apos;UnsetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.124'
      name = 'sendingInstant'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    555
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.125'
      name = 'processingDuration'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    75
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.126'
      name = 'sequenceNumber'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    1.2.4
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.127'
```

```
      name = 'operation'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#RaiseGroungFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.128'
      name = 'operationNameAsText'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    &apos;RaiseGroungFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.129'
      name = 'minProcessingDuration'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    75
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.130'
      name = 'sendingInstant'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    600
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.131'
      name = 'processingDuration'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.132'
      name = 'sequenceNumber'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    1.2.4.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.133'
      name = 'operation'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#SetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.134'
      name = 'operationNameAsText'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    &apos;SetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.135'
      name = 'sendingInstant'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    685
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.136'
```

```
      name = 'processingDuration'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    139
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.137'
      name = 'sequenceNumber'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    1.3
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.138'
      name = 'operation'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeleteAnnouncements#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.139'
      name = 'operationNameAsText'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    &apos;DeleteAnnouncements():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.140'
      name = 'minProcessingDuration'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    45
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.141'
      name = 'sendingInstant'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    713
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.142'
      name = 'processingDuration'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    101
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.143'
      name = 'sequenceNumber'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    1.3.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.144'
      name = 'operation'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeletePossibleGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.145'
```

```
      name = 'operationNameAsText'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    &amp;apos;DeletePossibleGroundFault():void&amp;apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.146'
      name = 'minProcessingDuration'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    87
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.147'
      name = 'sendingInstant'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    750
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.148'
      name = 'processingDuration'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.149'
      name = 'sequenceNumber'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    1.3.1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.150'
      name = 'operation'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    &amp;lt;oiref:java#Member#CabDisplay#GetDisplay#(##)#:oiref&amp;gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.151'
      name = 'operationNameAsText'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    &amp;apos;GetDisplay():void&amp;apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.152'
      name = 'sendingInstant'
      modelElement = 'G.42'>
 <UML:TaggedValue.dataValue>
    784
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.153'
      name = 'processingDuration'
      modelElement = 'G.42'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.154'
```

```
      name = 'sequenceNumber'
      modelElement = 'G.42'>
 <UML:TaggedValue.dataValue>
    1.3.1.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.155'
      name = 'operation'
      modelElement = 'G.42'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#UnsetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.156'
      name = 'operationNameAsText'
      modelElement = 'G.42'>
 <UML:TaggedValue.dataValue>
    &apos;UnsetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.157'
      name = 'sendingInstant'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    115
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.158'
      name = 'processingDuration'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    719
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.159'
      name = 'sequenceNumber'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.160'
      name = 'operation'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#HeadMeetsBeacon#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.161'
      name = 'operationNameAsText'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    &apos;HeadMeetsBeacon():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.162'
      name = 'minProcessingDuration'
      modelElement = 'G.26'>
 <UML:TaggedValue.dataValue>
    180
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.163'
```

```
      name = 'sendingInstant'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    124
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.164'
      name = 'processingDuration'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    151
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.165'
      name = 'sequenceNumber'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.166'
      name = 'operation'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#AnnounceSpeedRestriction#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.167'
      name = 'operationNameAsText'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    &apos;AnnounceSpeedRestriction():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.168'
      name = 'minProcessingDuration'
      modelElement = 'G.27'>
 <UML:TaggedValue.dataValue>
    30
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.169'
      name = 'sendingInstant'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    160
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.170'
      name = 'processingDuration'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    105
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.171'
      name = 'sequenceNumber'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    1.1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.172'
```

```
      name = 'minProcessingDuration'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
     42
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.173'
      name = 'operation'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeletePossibleGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.174'
      name = 'operationNameAsText'
      modelElement = 'G.30'>
 <UML:TaggedValue.dataValue>
    &apos;DeletePossibleGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.175'
      name = 'sendingInstant'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    210
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.176'
      name = 'processingDuration'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.177'
      name = 'sequenceNumber'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    1.1.1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.178'
      name = 'operation'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#GetDisplay#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.179'
      name = 'operationNameAsText'
      modelElement = 'G.31'>
 <UML:TaggedValue.dataValue>
    &apos;GetDisplay():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.180'
      name = 'sendingInstant'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    235
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.181'
```

```
      name = 'processingDuration'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.182'
      name = 'sequenceNumber'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    1.1.1.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.183'
      name = 'operation'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#UnsetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.184'
      name = 'operationNameAsText'
      modelElement = 'G.32'>
 <UML:TaggedValue.dataValue>
    &apos;UnsetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.185'
      name = 'sendingInstant'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    306
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.186'
      name = 'processingDuration'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    334
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.187'
      name = 'sequenceNumber'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    1.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.188'
      name = 'operation'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#AddSpeedRestriction#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.189'
      name = 'operationNameAsText'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    &apos;AddSpeedRestriction():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.190'
```

```
      name = 'minProcessingDuration'
      modelElement = 'G.28'>
 <UML:TaggedValue.dataValue>
    114
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.191'
      name = 'sendingInstant'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    350
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.192'
      name = 'processingDuration'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.193'
      name = 'sequenceNumber'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    1.2.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.194'
      name = 'operation'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#AnnounceBeacon#GetTargetSpeed#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.195'
      name = 'operationNameAsText'
      modelElement = 'G.33'>
 <UML:TaggedValue.dataValue>
    &apos;GetTargetSpeed():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.196'
      name = 'sendingInstant'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    390
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.197'
      name = 'processingDuration'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.198'
      name = 'sequenceNumber'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    1.2.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.199'
```

```
      name = 'operation'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#LimitBeacon#SetSpeedRestriction#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.200'
      name = 'operationNameAsText'
      modelElement = 'G.36'>
 <UML:TaggedValue.dataValue>
    &apos;SetSpeedRestriction():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.201'
      name = 'sendingInstant'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    415
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.202'
      name = 'processingDuration'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    100
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.203'
      name = 'sequenceNumber'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    1.2.3
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.204'
      name = 'operation'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeletePossibleGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.205'
      name = 'operationNameAsText'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    &apos;DeletePossibleGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.206'
      name = 'minProcessingDuration'
      modelElement = 'G.37'>
 <UML:TaggedValue.dataValue>
    35
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.207'
      name = 'sendingInstant'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    460
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.208'
```

```
      name = 'processingDuration'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.209'
      name = 'sequenceNumber'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    1.2.3.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.210'
      name = 'operation'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#GetDisplay#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.211'
      name = 'operationNameAsText'
      modelElement = 'G.38'>
 <UML:TaggedValue.dataValue>
    &apos;GetDisplay():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.212'
      name = 'sendingInstant'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    485
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.213'
      name = 'processingDuration'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.214'
      name = 'sequenceNumber'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    1.2.3.2
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.215'
      name = 'operation'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#UnsetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.216'
      name = 'operationNameAsText'
      modelElement = 'G.39'>
 <UML:TaggedValue.dataValue>
    &apos;UnsetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.217'
```

```
      name = 'sendingInstant'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    555
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.218'
      name = 'processingDuration'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    75
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.219'
      name = 'sequenceNumber'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    1.2.4
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.220'
      name = 'operation'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#RaiseGroungFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.221'
      name = 'operationNameAsText'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    &apos;RaiseGroungFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.222'
      name = 'minProcessingDuration'
      modelElement = 'G.34'>
 <UML:TaggedValue.dataValue>
    75
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.223'
      name = 'sendingInstant'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    600
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.224'
      name = 'processingDuration'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.225'
      name = 'sequenceNumber'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    1.2.4.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.226'
```

```
      name = 'operation'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#SetGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.227'
      name = 'operationNameAsText'
      modelElement = 'G.35'>
 <UML:TaggedValue.dataValue>
    &apos;SetGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.228'
      name = 'sendingInstant'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    685
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.229'
      name = 'processingDuration'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    139
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.230'
      name = 'sequenceNumber'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    1.3
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.231'
      name = 'operation'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeleteAnnouncements#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.232'
      name = 'operationNameAsText'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    &apos;DeleteAnnouncements():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.233'
      name = 'minProcessingDuration'
      modelElement = 'G.29'>
 <UML:TaggedValue.dataValue>
    45
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.234'
      name = 'sendingInstant'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    713
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.235'
```

```
      name = 'processingDuration'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    101
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.236'
      name = 'sequenceNumber'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    1.3.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.237'
      name = 'operation'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CSL#DeletePossibleGroundFault#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.238'
      name = 'operationNameAsText'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    &apos;DeletePossibleGroundFault():void&apos;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.239'
      name = 'minProcessingDuration'
      modelElement = 'G.40'>
 <UML:TaggedValue.dataValue>
    87
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.240'
      name = 'sendingInstant'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    750
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.241'
      name = 'processingDuration'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    20
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.242'
      name = 'sequenceNumber'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    1.3.1.1
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.243'
      name = 'operation'
      modelElement = 'G.41'>
 <UML:TaggedValue.dataValue>
    &lt;oiref:java#Member#CabDisplay#GetDisplay#(##)#:oiref&gt;
 </UML:TaggedValue.dataValue>
</UML:TaggedValue>
<UML:TaggedValue xmi.id = 'XX.244'
```

```
        name = 'operationNameAsText'
        modelElement = 'G.41'>
    <UML:TaggedValue.dataValue>
        &apos;GetDisplay():void&apos;
    </UML:TaggedValue.dataValue>
   </UML:TaggedValue>
   <UML:TaggedValue xmi.id = 'XX.245'
        name = 'sendingInstant'
        modelElement = 'G.42'>
    <UML:TaggedValue.dataValue>
        784
    </UML:TaggedValue.dataValue>
   </UML:TaggedValue>
   <UML:TaggedValue xmi.id = 'XX.246'
        name = 'processingDuration'
        modelElement = 'G.42'>
    <UML:TaggedValue.dataValue>
        20
    </UML:TaggedValue.dataValue>
   </UML:TaggedValue>
   <UML:TaggedValue xmi.id = 'XX.247'
        name = 'sequenceNumber'
        modelElement = 'G.42'>
    <UML:TaggedValue.dataValue>
        1.3.1.2
    </UML:TaggedValue.dataValue>
   </UML:TaggedValue>
   <UML:TaggedValue xmi.id = 'XX.248'
        name = 'operation'
        modelElement = 'G.42'>
    <UML:TaggedValue.dataValue>
        &lt;oiref:java#Member#CabDisplay#UnsetGroundFault#(##)#:oiref&gt;
    </UML:TaggedValue.dataValue>
   </UML:TaggedValue>
   <UML:TaggedValue xmi.id = 'XX.249'
        name = 'operationNameAsText'
        modelElement = 'G.42'>
    <UML:TaggedValue.dataValue>
        &apos;UnsetGroundFault():void&apos;
    </UML:TaggedValue.dataValue>
   </UML:TaggedValue>
  </XMI.content>
</XMI>
```

Appendix-IV

Integration Test Paths for the *HeadMeetsBeacon* Event

This appendix lists the test paths for the *HeadMeetsBeacon* event of CSLaM case study of Chapter 7 under All-Path coverage.

T_1: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_2: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_3: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_4: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_5: $(m_1 \rightarrow CSL{:}p_2(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_6: $(m_1 \rightarrow CSL{:}p_2(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_7: $(m_1 \rightarrow CSL{:}p_2(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_8: $(m_1 \rightarrow CSL{:}p_2(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_9: $(m_1 \rightarrow CSL{:}p_3(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_{10}: $(m_1 \rightarrow CSL{:}p_3(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_{11}: $(m_1 \rightarrow CSL{:}p_3(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_{12}: $(m_1 \rightarrow CSL{:}p_3(m_2 \rightarrow CSL{:}p_5(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19}))))$

T_{13}: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{12}(m_{11} \rightarrow CabDisplay{:}p_{19})$
$(m_{12} \rightarrow CabDisplay{:}p_{19}))))$

T_{14}: $(m_1 \rightarrow CSL{:}p_1(m_2 \rightarrow CSL{:}p_4(m_5 \rightarrow CSL{:}p_{13}(m_{11} \rightarrow CabDisplay{:}p_{19})$

$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{15}: $(m_1 \rightarrow CSL:p_1(m_2 \rightarrow CSL:p_5(m_5 \rightarrow CSL:p_{12}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{16}: $(m_1 \rightarrow CSL:p_1(m_2 \rightarrow CSL:p_5(m_5 \rightarrow CSL:p_{13}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{17}: $(m_1 \rightarrow CSL:p_2(m_2 \rightarrow CSL:p_4(m_5 \rightarrow CSL:p_{12}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{18}: $(m_1 \rightarrow CSL:p_2(m_2 \rightarrow CSL:p_4(m_5 \rightarrow CSL:p_{13}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{19}: $(m_1 \rightarrow CSL:p_2(m_2 \rightarrow CSL:p_5(m_5 \rightarrow CSL:p_{12}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{20}: $(m_1 \rightarrow CSL:p_2(m_2 \rightarrow CSL:p_5(m_5 \rightarrow CSL:p_{13}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{21}: $(m_1 \rightarrow CSL:p_3(m_2 \rightarrow CSL:p_4(m_5 \rightarrow CSL:p_{12}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{22}: $(m_1 \rightarrow CSL:p_3(m_2 \rightarrow CSL:p_4(m_5 \rightarrow CSL:p_{13}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{23}: $(m_1 \rightarrow CSL:p_3(m_2 \rightarrow CSL:p_5(m_5 \rightarrow CSL:p_{12}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{24}: $(m_1 \rightarrow CSL:p_3(m_2 \rightarrow CSL:p_5(m_5 \rightarrow CSL:p_{13}(m_{11} \rightarrow CabDisplay:p_{19})$
$(m_{12} \rightarrow CabDisplay:p_{19})))$

T_{25}: $(m_1 \rightarrow CSL:p_1(m_3 \rightarrow CSL:p_6(m_6 \rightarrow AnnounceBeacon:p_{14})$
$(m_7 \rightarrow LimitBeacon:p_{15})(m_8 \rightarrow CSL:p_{16}(m_{13} \rightarrow CabDisplay:p_{19})))$

T_{26}: $(m_1 \rightarrow CSL:p_1(m_3 \rightarrow CSL:p_6(m_6 \rightarrow AnnounceBeacon:p_{14})$
$(m_7 \rightarrow LimitBeacon:p_{15})(m_8 \rightarrow CSL:p_{17}(m_{13} \rightarrow CabDisplay:p_{19})))$

T_{27}: $(m_1 \rightarrow CSL:p_1(m_3 \rightarrow CSL:p_7(m_6 \rightarrow AnnounceBeacon:p_{14})$
$(m_7 \rightarrow LimitBeacon:p_{15})(m_8 \rightarrow CSL:p_{16}(m_{13} \rightarrow CabDisplay:p_{19})))$

T_{28}: $(m_1 \rightarrow CSL:p_1(m_3 \rightarrow CSL:p_7(m_6 \rightarrow AnnounceBeacon:p_{14})$
$(m_7 \rightarrow LimitBeacon:p_{15})(m_8 \rightarrow CSL:p_{17}(m_{13} \rightarrow CabDisplay:p_{19})))$

T_{29}: $(m_1 \rightarrow CSL:p_1(m_3 \rightarrow CSL:p_8(m_6 \rightarrow AnnounceBeacon:p_{14})$
$(m_7 \rightarrow LimitBeacon:p_{15})(m_8 \rightarrow CSL:p_{16}(m_{13} \rightarrow CabDisplay:p_{19})))$

T_{30}: $(m_1 \rightarrow CSL:p_1(m_3 \rightarrow CSL:p_8(m_6 \rightarrow AnnounceBeacon:p_{14})$
$(m_7 \rightarrow LimitBeacon:p_{15})(m_8 \rightarrow CSL:p_{17}(m_{13} \rightarrow CabDisplay:p_{19})))$

T_{31}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{32}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{33}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{34}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{35}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{36}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{37}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{38}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{39}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{40}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{41}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{42}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{43}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{44}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{45}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{46}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19}))))$

T_{47}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$

$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})))$

T_{48}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})))$

T_{49}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{50}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{51}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{52}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{53}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{54}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{55}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{56}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{57}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{58}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{59}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})$
$(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})$
$(m_{14} \rightarrow CabDisplay{:}p_{19})))$

T_{60}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{61}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{62}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{63}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{64}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{65}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{66}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_6(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{67}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{68}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_7(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{69}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{70}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_8(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{71}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{16}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{72}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_9(m_6 \rightarrow AnnounceBeacon{:}p_{14})(m_7 \rightarrow LimitBeacon{:}p_{15})(m_8 \rightarrow CSL{:}p_{17}(m_{13} \rightarrow CabDisplay{:}p_{19})(m_{14} \rightarrow CabDisplay{:}p_{19}))))$

T_{73}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_6(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{74}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_6(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{75}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_7(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{76}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_7(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{77}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_8(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{78}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_8(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{79}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_9(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{80}: $(m_1 \rightarrow CSL{:}p_1(m_3 \rightarrow CSL{:}p_9(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{81}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_6(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{82}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_6(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{83}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_7(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{84}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_7(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{85}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_8(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{86}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_8(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{87}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_9(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{88}: $(m_1 \rightarrow CSL{:}p_2(m_3 \rightarrow CSL{:}p_9(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{89}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_6(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{90}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_6(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{91}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_7(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{92}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_7(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{93}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_8(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{94}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_8(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{95}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_9(m_9 \rightarrow CSL{:}p_{18}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{96}: $(m_1 \rightarrow CSL{:}p_3(m_3 \rightarrow CSL{:}p_9(m_9 \rightarrow CSL{:}p_{13}(m_{15} \rightarrow CabDisplay{:}p_{19}))))$

T_{97}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{98}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{99}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{100}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{101}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{102}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{103}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{104}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{105}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{106}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{107}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{108}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19}))))$

T_{109}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{110}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{111}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{112}: $(m_1 \rightarrow CSL{:}p_1(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{113}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{114}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{115}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{116}: $(m_1 \rightarrow CSL{:}p_2(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{117}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{118}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{10}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{119}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{12}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$

T_{120}: $(m_1 \rightarrow CSL{:}p_3(m_4 \rightarrow CSL{:}p_{11}(m_{10} \rightarrow CSL{:}p_{13}(m_{16} \rightarrow CabDisplay{:}p_{19})$
$(m_{17} \rightarrow CabDisplay{:}p_{19}))))$