DESIGN LEVEL COUPLING METRICS FOR UML MODELS

BY

SOHEL KHAN

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

JANUARY 2004

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

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **SOHEL KHAN** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Thesis Committee

Dr. Jarallah AlGhamdi
(Chairman)

Dr. Krishna Rao
(Member)

Dr. Paul Manuel
(Member)

Dr. Kanaan Faisal
(Department Chairman)

Prof. Osama Ahmed Jannadi
(Dean of Graduate Studies)

10 - 3 - 2004
Date

Acknowledgments

All thanks are due Allah first and foremost for his countless blessings. Acknowledgement is due to KFUPM and the ICS Department for supporting this research.

I was honored and privileged to have Dr. Jarallah AlGhamdi as my advisor. I feel deeply indebted to him for shaping this research with his extensive knowledge and experience. I am thankful to him for bearing with patience my every mistake, correcting it and encouraging at every step. I also wish to thank my thesis committee members, Dr. Paul Manuel and Dr. Krishna Rao for their help, support, and contributions.

My thankful admiration goes to my colleague, pair programmer Rufai for positively contributing to my research through constructive criticism and discussions. Many of my ideas would have remained shallow and abstract but for the insightful discussions and questions at Software Metrics Research Group (SMRG) and Dr. Moataz Ahmed for motivating my research with valuable ideas and rich experience. My thanks go to all members of the SMRG. Thanks to all my friends and colleagues at KFUPM who provided wholesome companionship throughout my studies at KFUPM.

Above all I thank my family for all their support and love throughout my MS studies.

Table of Contents

List of Figures

List of Tables

Thesis Abstract

NAME: SOHEL KHAN

TITLE: DESIGN LEVEL COUPLING METRICS FOR UML MODELS

MAJOR FIELD: COMPUTER SCIENCE

DATE OF DEGREE: JANURAY 2004

This thesis proposes an approach for computing coupling metrics from object oriented UML models stored in XMI format. The coupling metrics are computed from the package, class and sequence diagrams in the UML model. A metric tool is developed for extracting the design level information and computing the metrics. Our experimental results are discussed and compared with results from source level metrics. Results show that design level coupling metrics are consistent with those computed from the source code.

خلاصة الرسالة

الاســـــم : صهيل خان

عنوان الدراسة : قياسات الاقتران على مستوى التصميم لنماذج UML

التخصص : علوم الحاسب الآلي

تاريخ التخرج : يناير 2004

يقترح هذا البحث طريقة حسابية لقياس الاقتران (Coupling) من نماذج UML الكائنية والمحفوظة بصيغة XMI. تُحسب قياسات الاقتران من حزمة برامج ومن فئة برمجية ومن مخططات متسلسلة في نموذج UML. تم فى هذا البحث تطوير أداة قياس لاستخلاص المعلومات من التصميم وتقوم هذه الأداة بعد ذلك بحساب القياسات. وفي نهاية البحث تمت مناقشة ومقارنة النتائج الاختبارية مع القياسات المستخلصة من البرامج (تطبيقات التصميم). وقد أظهرت النتائج أن قياسات الاقتران على مستوى التصميم تتوافق مع تلك النتائج المستخلصة من البرامج.

درجة الماجستير في العلوم

جامعة الملك فهد لبترول والمعادن

يناير 2004

Chapter 1

Introduction

Software Metrics can help to understand measure, analyze, control and improve software product and process attributes. Coupling is a software product attribute that is defined as the degree of inter-connection or dependency between two or more units of software (a unit may be a class, module, package, or system). Because coupling can give insight into the complexity, error density, error propagation and quality of software, it has been extensively studied in the software metrics literature and various metrics have been proposed in the literature to measure this attribute. Coupling is mainly a design-level metrics. Since the Unified Modeling Language (UML) is widely used for expressing design artifacts in Object Oriented Software Development (OOSD), it would be worthwhile to assess coupling directly from UML models. In this research we investigated that whether coupling could be derived at design level of software and propose a suite of coupling metrics that can be computed for UML designs (package, class and sequence/collaboration diagrams) stored in XMI format. Hence we get all the advantages of UML like early availability, as well as coupling metric like complexity estimate, effort prediction, error proneness and understandability. In the remainder of the Chapter we will discuss the

concept of coupling, coupling in object oriented programs and motivation for design level coupling.

1.1 Coupling

Coupling is an attribute of software that is defined as the degree of inter-connection or dependency between two or more units (a unit may be a class, module, package, system etc) of software [31]. Various metrics are proposed in literature to measure this attribute.

1.1.1 Analogies of Coupling

Different disciplines of science have benefited immensely from intellectual exchange of ideas among them. Hence we explore how the concept of coupling came to software engineering from other disciplines. Literally the word "couple" refers to two items of same kind, or a link that binds two things together and informally used for a small number of things joined together ("a couple of days") [1].

Following are some interesting analogies of coupling that exist in other disciplines.

- Word "couple" was used in Physics to mean two forces acting in parallel but opposite in direction to cause rotation [2].

- Its verb form "coupling" came into Electronics to refer to the interference cause by electrical signals from one part of circuit to another [3].
- Optical coupling in optical circuits is defined as the transmission of light energy from one source to other [4].

In electronic circuits though coupling is sometimes considered harmful, it can also be desirable in some cases. For example it is harmful in communication circuits but useful in measuring circuits to measure attributes of other circuits [3]. Optical coupling is a desirable feature in optical circuits and efforts continue to make effective use of it [4].

Its earliest usage in software engineering dates back to 1974 by Stevens et. al. [31] in one of their most cited works on structured programming titled "Structured Design". They defined coupling as the complexity of communication between different modules.

1.2 Coupling in Object Oriented Programs

Initially the coupling metrics were designed and used for measuring programs based on conventional structured programming paradigm [32]. Coupling was measured from the extent to which one function uses other

function or its members. Thus types of connections that could occur are methods call or attribute use.

With the advent of Object Oriented Software Development, coupling metrics were proposed specially geared towards measuring coupling in object-oriented programs [33]. Object technology raised the level of abstraction from functional units to objects and classes. Hence for measuring coupling of object oriented software, different types of connections that can occur between classes has to be considered.

1.3 Motivation

The process of raising level of abstraction continues as software engineers try to decrease the complexity of programs and enable the development of quality software for large real life applications [5][6][7][8]. This explains the emphasis today on design methodologies, notations to express design, design patterns in the software engineering community [9]. As software engineers realized importance of design and architecture, different methodologies sprang up to support their applications. Unified Modeling Language (UML) [18] has become the defacto standard for expressing software requirements, design and process artifacts. There is a dearth of useful metrics for measuring UML models in general and coupling metrics in particular, although there have been some elementary metrics proposed recently. However these are

mostly elementary measures e.g., UML numbers of classes, number of messages [11][12]. Hence the important issues that this thesis addresses are namely:

- Can coupling be derived from design?
- How is it different than Object technology?
- What could be its application?

Coupling is a major property of design that signifies the estimate of dependency or relationship that each module has with other modules of system and metrics have been proposed to measure different types of coupling. But there are few works that try to measure coupling for UML models.

Measuring coupling at design level offer the following advantages:

- Early detection of flaws.
- They can help to understand, assess software architecture in the design stage and choose between alternatives.
- Gain insight into the development process when compared with source level coupling .
- Measuring coupling of UML artifacts can be an important step forward in understanding and improving their use.

1.4 Main Contributions

The main contributions of this thesis work are the following:

- Conducting an extensive critical survey of existing coupling metrics, UML metrics and XMI processing.
- Demonstrating that information required for computing complex metrics like coupling metrics can be gathered from UML diagrams.
- Proposing high level coupling metrics for UML models.
- Designing and building a UML Metric tool for parsing UML models stored in XMI files and computing the coupling metrics.
- Conducting case studies to show that the design level coupling metric convey similar information as conveyed by code level coupling metrics.
- Exploring the relationship of UML coupling metrics to other design level metrics.
- Introducing an approach of using design level and code level metrics to gain insight into development process and identify the problems introduced at the coding level.

1.5 Organization of Thesis

The rest of this thesis is organized as follows. Chapter 2 gives an extensive literature survey of coupling related work. Chapter 3 presents the taxonomy for coupling. Chapter 4 presents UML coupling metrics and approach for measuring them from XMI. Chapter 5 explains the implementation of tool that we designed and developed for measuring coupling of UML models. Chapter 6 gives the theoretical validation and result of experiments for empirical validation of UML coupling metrics. Chapter 7 gives the conclusion and further work.

Chapter 2

Literature Survey

In this chapter, we discuss related work on coupling metrics and modeling concepts. The classical coupling metric related work is divided into three major parts, section 2.2 discuss the work of pioneers who initiated the concept of coupling. Section 2.3 covers different coupling metric proposed in literature. Section 2.4 discusses dynamic coupling metrics. Section 2.5 discusses the metrics that have been proposed and implemented for UML. Section 2.6 presents an overview of modeling concepts.

2.1 Coupling

Parnas [30] in 1972 in a seminal paper titled "On the criteria to be used in decomposing systems into modules" proposed the criteria of information hiding to be used for system decomposition or modularization of large systems. He discussed on methods for decomposition of functions and need of information hiding between them. The same concept was later refined and used to come up with design rules for well-formed classes in object-oriented design like low coupling and high cohesion, although Parnas did not use the term "coupling" in his work. It was also one of the motivations for our work on package level coupling discussed in Section 5.5.

The earliest reference we could find in the literature to the term "coupling in software engineering" is by Stevens, Myers and Constantine in 1974 [31]. In his significant paper titled "Structured Design" authors have suggested several design guidelines under the discipline of structured design for developing large software systems. These guidelines were claimed to increase the quality attributes such as simplicity, modifiability, predictability and observability of the system. The concept of coupling was introduced in this work to account for connections between modules. The module was defined as the subprogram, procedure or function. Although the concepts and design guidelines introduced in that paper were based on procedural paradigm but the concepts were more generic and relevant for other paradigms like Object Oriented Design. Hence the fundamental basis for many of the coupling metrics (and other metrics for design quality such as cohesiveness) can be witnessed in the paper. They defined coupling on an ordinal scale (low, high) depending on the complexity of connection, strength of connection and the complexity of message involved in that connection. They classified coupling into data and control coupling. Data coupling occurs when data was shared between modules and control coupling occurs when the connection affects the control or execution of other module. Control coupling was identified as more severe form of coupling than data coupling and guidelines called structured design techniques were

discussed on the ways to minimize control coupling in specific and coupling in general and improve the design quality. Some of the techniques and concepts introduced in paper like "scope of control", "scope of effect", "highest level of abstraction", "isolation", "reducing dependencies" can be seen as precursors to the birth of object oriented design. But the authors did not introduce any metric as such for measuring coupling nor did they conduct any experiments towards the same.

Myers 1978 classified coupling into six types on an ordinal scale, from the most preferable type of "no coupling" to content coupling that was worse for a design. These types were:

- **No Coupling**: The two modules are not connected by any means
- **Data Coupling**: Direct communication between modules with all interface data being homogenous data items
- **Stamp Coupling**: Two modules coupled by the function arguments of a call.
- **Control Coupling**: One module controls the execution sequence or logic of other module.
- **External**: Two or more modules reference a homogenous global data item.

- **Common Coupling**: Two modules coupled with global shared data.
- **Content Coupling**: One module refers to the internal data of another module.

2.2 Object Oriented Coupling

Coad and Yourdon [33] were the first to classify coupling into interaction and inheritance coupling as criteria for the design in their work on structured design and object-oriented design.

- **Interaction coupling**: This was defined as the coupling that results from the exchange of messages between two classes. They proposed that a message connection should normally not have more than three parameters.
- **Inheritance Coupling**: This was defined as the interconnection between generalizations and specializations. They cautioned against the overuse of Inheritance.

Berard [34] classified coupling into necessary and unnecessary types. He was also the first to introduce the concept of **Object Coupling**. "Unnecessary object coupling needlessly decreases the reusability of the coupled objects. Unnecessary object coupling also increases the chances of

system corruption when changes are made to one or more of the coupled objects"[34]

Chidamber & Kemerer [36] were one of the first to propose a formal coupling metric for object-oriented systems, in 1994. They defined their coupling metric **CBO** (coupling between objects) as number of variables or methods a class uses from other class. This use count is counted on both sides of the relationship. They validated their metric theoretically using Weyukers [114] formal properties that complexity metric should have like Monotonicity, interaction, non coarseness, non uniqueness and permutation. They also validated their metrics empirically to explore their relationship with external quality attributes of software. They found that high coupling suggests complex classes. One of the limitations of CBO was that frequency of use, use of objects and event handling were ignored. Polymorphic calls were accounted in all possible classes.

Eder [43] in 1994 classified coupling into 3 major types Interaction, Component and Inheritance. For Interaction coupling they used the same definition of Myers. They classified Inheritance coupling on an ordinal scale from best to worst as:

- **Modification Coupling:** It was further divided as signature modification and implementation modification. In the former

implementation as well as signature of an inherited method is changed whereas in the latter only implementation is changed.

- **Refinement Coupling** where the inherited methods are not completely modified but only refined. Hence on the same lines this was also divided into:
    - **Signature refinement Coupling** where the signature of inherited method is changed but **semantics** remain unchanged.
    - **Implementation Refinement Coupling** where the implementation of inherited methods is refined with same semantics.
- **Extension Coupling** is the last type in inheritance coupling where a method or instance variable is added to child class, and it is most desirable type of coupling (except nil) in a system.

Interface Coupling was defined as the coupling that exists between classes if one class is the domain of instance variable, parameter, parameter of a method invoked within the class, or local variable of another class. This was said to be a measure of how explicit the coupling is between classes. It was again classified on an ordinal scale from worst to best as follows:

- **Hidden Coupling:** when an object of other class is used in implementation of a method not explicitly but as a return value of a call to another method.
- **Scattered Coupling:** when another class is used as domain of instance variable or a local variable.
- **Specified Coupling:** when it is specified in the specification of the class that another class is used as domain of one of the instance variable.

Although they explored the concept of coupling in detail and the work extended the understanding of coupling for object oriented system, but it cannot be called as coupling metrics. This is because most of the definitions of different types of coupling do not lend themselves easily for objective and automatic extraction from code. Concepts like semantics of a method are very difficult to automatically verify.

Hitz and Montazeri [44] defined two levels of coupling, Class level coupling and Object level coupling. Although object coupling was introduced earlier, they proposed a method to estimate it. They used the concept of "stability of class" to define the measure, but stability of class was itself left unmeasured. In absence of any objective measure for stability of a class it is difficult to measure this metric. Another issue with their object coupling is that they

claim that it captures dynamic coupling, from run time behavior of system, but they estimate the dynamic coupling from static code. In contrast the two approaches that we discuss in next chapter 2.3 for dynamic coupling are better estimates of dynamic coupling.

Troy and Zweben [48] proposed 24 measures to analyze the modularity, the size, the complexity, the cohesion and the coupling of a software system. The basic division of software (complexity) measures into inter-modular and intra-modular components and the specific conceptual measures of coupling and cohesion are based on a work of Stevens et. al. [31].

In the above survey we did not include the coupling measures that differ in some minor ways with the above metrics.

2.3 Dynamic coupling

Erich [59] mentions some limitations of static measures in his paper. By static measures he means the measures derived from source code. Limitations cited are that Static measures depend on static syntactic features of a programming language. This information does not show the dynamic behavior accurately. To know the actual dynamic execution behavior from static code is hard as it involves excessive logical or semantic analysis. They propose a test environment to capture the dynamic behavior from the

program execution profile. But their work was limited to modeling and they did not implement the program monitor nor did they compute any dynamic coupling metric.

In contrast Henrique et. al. [60] proposed and implemented an approach for estimating the dynamic coupling connections from the use case in the analysis and design phase. They also discussed the problem of Application partitioning, especially in distributed environment to minimize communication as an NP-Complete problem. They have discussed some of the visual tools that have been developed to assist the designer in portioning. They propose a linear algorithm DCM (Dynamic Clustering Mechanism) that constructs clusters of classes that are highly dynamically coupled. In their approach first the user has to specify the frequencies for each use case with which that use case is expected to be used in the application. These frequencies are then percolated down to calculation of dynamic coupling between the classes that each use case represents. For each cluster of classes they calculate the interaction inside the cluster and interactions going outside the cluster.

Sherif et. al. [64] emphasized the significance of dynamic metrics for real time object oriented systems. They propose two dynamic coupling metrics Export and Import Object Coupling in terms of number of messages sent and

received by a class respectively from another class. They experimented with their metric using a real time application as a case study. Results show that static and dynamic coupling measures differ and offer different indications on quality of software.

Erik et.al. [68][69] also did a similar work on dynamic coupling, but they tried to empirically validate dynamic coupling metric as an indicator of change proneness of a class. The results do show a positive correlation between the two.

Aine [65] took a different approach to capture dynamic behavior. She used the JVMDI (JVM Debug Interface) to capture the dynamic method invocations from the class files of running programs. She used the java benchmark SPEC JVM 98 as a case study to show that dynamic coupling differs in some case from the static counterpart.

Gamma et. al. [103] states that an Object Oriented program's run time structure often bears little resemblance to its code structure.

In summary dynamic coupling metrics are important for following reasons:

- Can give accurate estimate of coupling at run time
- Can take into account Polymorphic invocations of object-oriented systems more accurately, which is not possible from static analysis

2.4 UML Metrics

Design level metrics based on UML has been classified as elementary metrics, composite metrics and higher level metrics.

- Elementary metrics are those that give a count of basic entities such as number of classes, attributes.
- Composite metrics are those that are based on elementary metrics and try to measure more abstract and complex measures like coupling, cohesion etc.
- Higher-level metrics are those that try to use the elementary and composite metrics to estimate or predict some external attribute of the system like size, maintenance effort, etc.

In the remainder of this section we describe the metrics that are proposed in each of these categories.

2.4.1 Elementary UML Metrics

Amador et. al. [84] have developed a tool to validate the quality of requirements by using heuristics to process the requirements specified in XML(Extensible Markup Language) format using XSLT(XSL Transformations) [23]. But they did not define any concrete metric, and that

is conceivably a difficult task owing to the nature of requirements specifications.

Kim et. al. [87] have proposed some basic UML metrics and incorporated them into their tool "UML Metrics Producer (UMP)" based on rational Rose. The metrics are counts of all basic elements of UML diagrams like class diagrams, use case and sequence diagrams. Hence they include most of other elementary metrics. The work lacks any validation effort.

Almost similar metrics were also proposed by Marcela [75].

Anh Le et. al. [80] used NSUML API for parsing UML class diagrams stored in XMI format and calculating twelve elementary metrics from them.

Trevor Paterson [78] also calculated some elementary metrics, but using XSLT.

All the elementary metrics are listed in Table 1.

Table 1: Elementary Metrics

Metric	Explanation	UML diagram	Reference
NC	Numbers of classes	Class diagrams	[12]
NA	Numbers of attributes	Class diagrams	[12]
NM	Numbers of methods	Class diagrams	[12]
Nassc	Numbers of associations	Class diagrams	[12]

Metric	Explanation	UML diagram	Reference
Nagg	Numbers of aggregations	Class diagrams	[12]
Ndep	Numbers of dependencies	Class diagrams	[12]
NgenH	Numbers of generalizations	Class diagrams	[12]
Max DIT	Maximum distance from class to root	Class diagrams	[12]
MaxHagg	Maximum distance from class to any leaf	Class diagrams	[12]
NEntryA	Number of entry actions	State Charts	[12]
NExitA	Number of exit actions	State Charts	[12]
NA	Number of activities	State Charts	[12]
NS	Number of states	State Charts	[12]
NT	Number of transitions	State Charts	[12]
Nact	Number of actors	Use cases	[12]
Nuse	Number of use cases	Use cases	[12]
NMessage	Number of messages	Sequence diagrams	[12]
NClassf	Number of Classifiers	Sequence diagrams	[12]

2.4.2 Composite UML Metrics

Marcela et. al. [86] have proposed complexity metrics for UML class diagrams. The metric is based on number of classes, methods, generalizations, dependencies, aggregations and associations in the class diagram.

Marcela et. al [83] derive some metrics from UML Statechart Diagrams. Again the metrics are counts of the statechart like transitions, exit action etc

and lack any theoretical basis to serve as an indicator of a useful property. They did try to assess the metrics as an indicator of understandability, but it lacked rigorous empirical validation, as the experimental objects were students of a course.

Table 2: Composite Metrics

Metric	Explanation	UML diagram	Definition	Ref.
AsvsC	Associations vs Classes	Class diagrams	$ASvsC = \left(\frac{n(assc)}{n(assc) + n(class)} \right)^2$	[83]
AgvsC	Aggregations vs Classes	Class diagrams	$AGvsC = \left(\frac{n(agg)}{n(agg) + n(class)} \right)^2$	[83]
DEPvsC	Dependencies vs Classes	Class diagrams	$DEPvsC = \left(\frac{n(dep)}{n(dep) + n(class)} \right)^2$	[83]
GevsC	Generalizations vs Classes	Class diagrams	$GEvsC = \left(\frac{n(gen)}{n(gen) + n(class)} \right)^2$	[83]
Mgh	Complexity of a generalization hierarchy Where $FLEAF_i$ is the number of leaf classes	Class diagrams	$FLEAF_i = \frac{N_i^{LEAF}}{N_i^C}$ $CJ_i = FLEAF_i - \frac{FLEAF_i}{ALLSUP_i}$	[83]
Mmi	Complexity of a generalization hierarchy due to multiple inheritance Where N^{EX} is	Class diagrams	$CMI_i = \frac{N_i^{EX}}{N_i^C}$ $M_{MI} = \sum_{i=1}^{n} CMI_i$	[83]

Metric	Explanation	UML diagram	Definition	Ref.
	the number of extra parents			
AvsC	Attribute vs Classes	Class diagrams	$AvsC = \left(\frac{n(att)}{n(att) + n(class)} \right)^2$	[83]
MvsC	Methods vs Classes	Class diagrams	$MvsC = \left(\frac{n(meth)}{n(meth) + n(class)} \right)^2$	[83]

2.4.3 Higher Level UML Metrics

Rufai [93] proposed a metric for structural similarity between two UML models represented by their class diagrams and implemented a tool UML Model Comparer for computing it.

Jukka et. al. [81] emphasized the lack of quality assurance methods for software architecture design. They implemented a method for extracting the pattern used in the architecture from the Class diagrams and Sequence diagrams. They implemented the algorithm for pattern mining problem using constraint satisfaction problem (CSP) approach and implement a tool MAISA (Metrics for Analysis and Improvement of Software Architectures) in java. It computes metrics predicting the quality attributes of the system and some other size metrics.

2.4.4 UML Metrics Tools

Peter et. al. [79] have worked on a tool OSMAT for capturing metrics from UML diagrams. The metrics calculated by OSMAT are mostly elementary

metrics from Rational petal files, but they do mention other metrics in their future plan.

Anh Le et. al.[80] implemented a UML metric tool based on NSUML API for parsing UML class diagrams stored in XMI format and calculating twelve elementary metrics from them.

Trevor Paterson [78] implemented a UML metric tool based on XSLT. It computes some elementary metrics. Their work is significant for XMI processing, but the metrics extracted are elementary.

SDMetrics [96] is a commercial UML metric tool that computes an exhaustive list of elementary metrics as well as some composite metrics.

2.5 Modeling Concepts

Models play a very important role in software engineering to reduce the complexity of application and enable designers to concentrate on the core concepts, relationships to create a design that can satisfy the customer requirements. Since our work is based on design models that are expressed in UML [18] we present a brief overview of related concepts like software modeling, Meta-Object Facility [21] (MOF), UML [18] (Unified Modeling Language) and XML Metadata Interchange Format [20] (XMI) in this section.

2.5.1 Software Modeling

Formally a model is a set of statements about some system under study (SUS) that may be used as a specification for an SUS, or for a class of SUS. In this case, a specific SUS is considered valid relative to this specification if no statement in the model is false for the SUS [13]. UML is a modeling language for information-processing systems (an SUS). Software design is not easy to define precisely because there are differing views on it. Subrata [15] classifies the concept into two schools of thought: one that views it as a blueprint for implementation and the other view of formalists who view programs as mathematical objects and design as an exercise in applied mathematics.

Going step higher above model we have "metamodel" that is a specification model for which the SUS being specified is itself model in a certain modeling language. It can also be defined as a language or vocabulary for defining the model. Thus UML Specification provides a metamodel of UML, as a set of statements about UML models that must not be violated by any valid UML model [13]. MOF [21] (Meta-Object Facility) is the OMG's metamodel specification for which the Information System being specified is itself a model in UML. Thus UML becomes the minimal reflexive metamodeling language for UML and MOF [13]. MDA (Model Driven Architecture) is an attempt of OMG to harness the strength of all the above

modeling concepts to come up with an approach for specifying functionality and behavior of a system, independent of method of implementation. Thus there is no need to repeat the process of modeling for every new implementation. "A complete MDA specification consists of a definitive platform-independent base UML model (PIM), plus one or more platform-specific models (PSM) and interface definition sets, each describing how the base model is implemented on a different middleware platform" [22]. The next sections will explain the OMG's specification of UML and XMI in some detail.

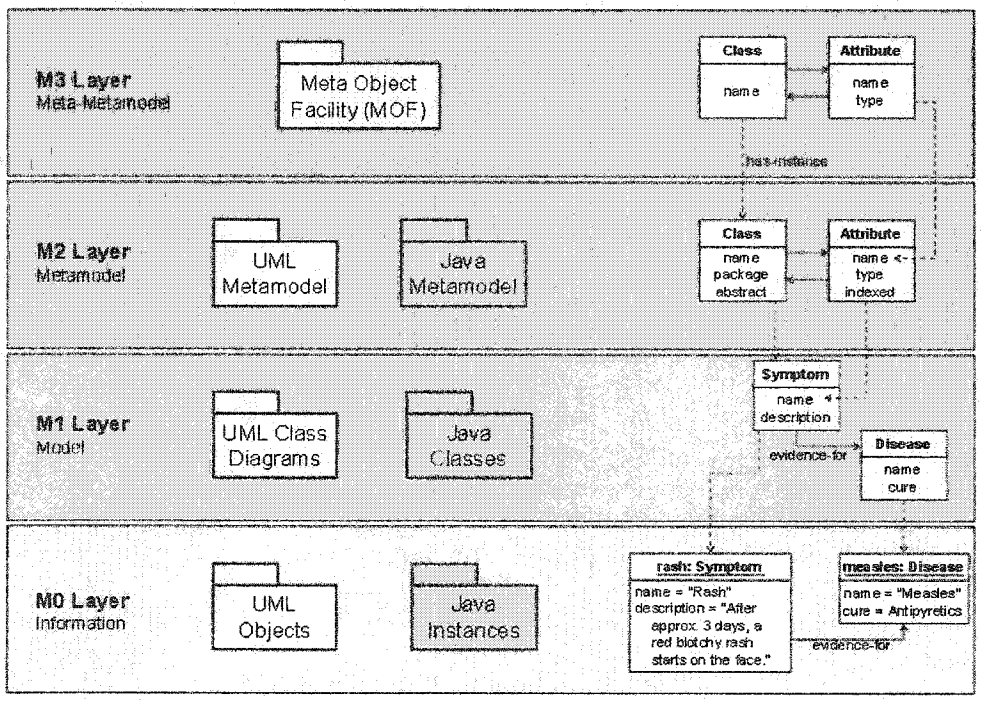

Figure 1:OMG Meta Model Architecture [97]

2.5.2 UML

The Unified Modeling Language (UML), proposed by OMG (Object Management Group) is now industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. Its success lies in the fact that it enjoys widespread industry support and methodology-independence. Regardless of the methodology used to perform analysis and design, UML can be used to express the results.

2.5.2.1 UML Semantics

The description that follows explains the semantics of three diagrams of UML (Package, Class and Sequence) that are relevant for this work, from its specification and the Rational Rose UML models available from OMG.

UML consists of three main packages at the highest level: Behavioral Elements, Foundation, and Model Management, as shown in package diagram of Figure 2.

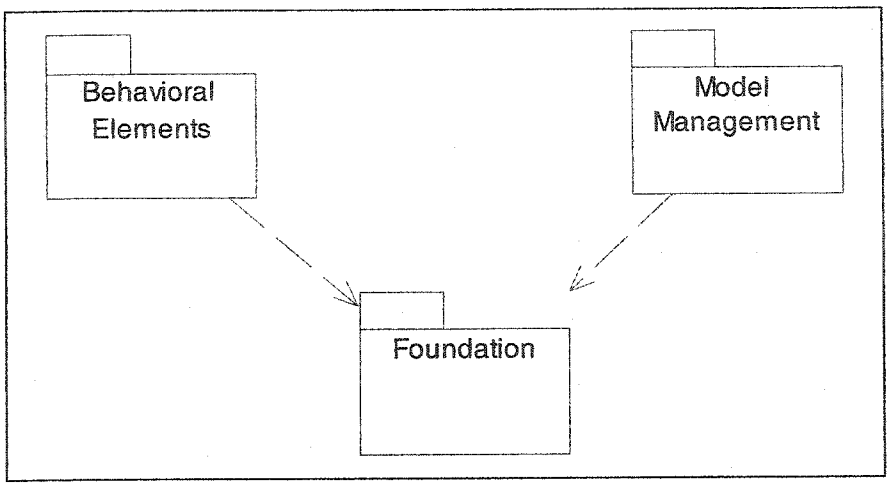

Figure 2: UML Main Packages

The Model Management package defines how the elements are organized in models, packages and systems as shown in Figure 3.

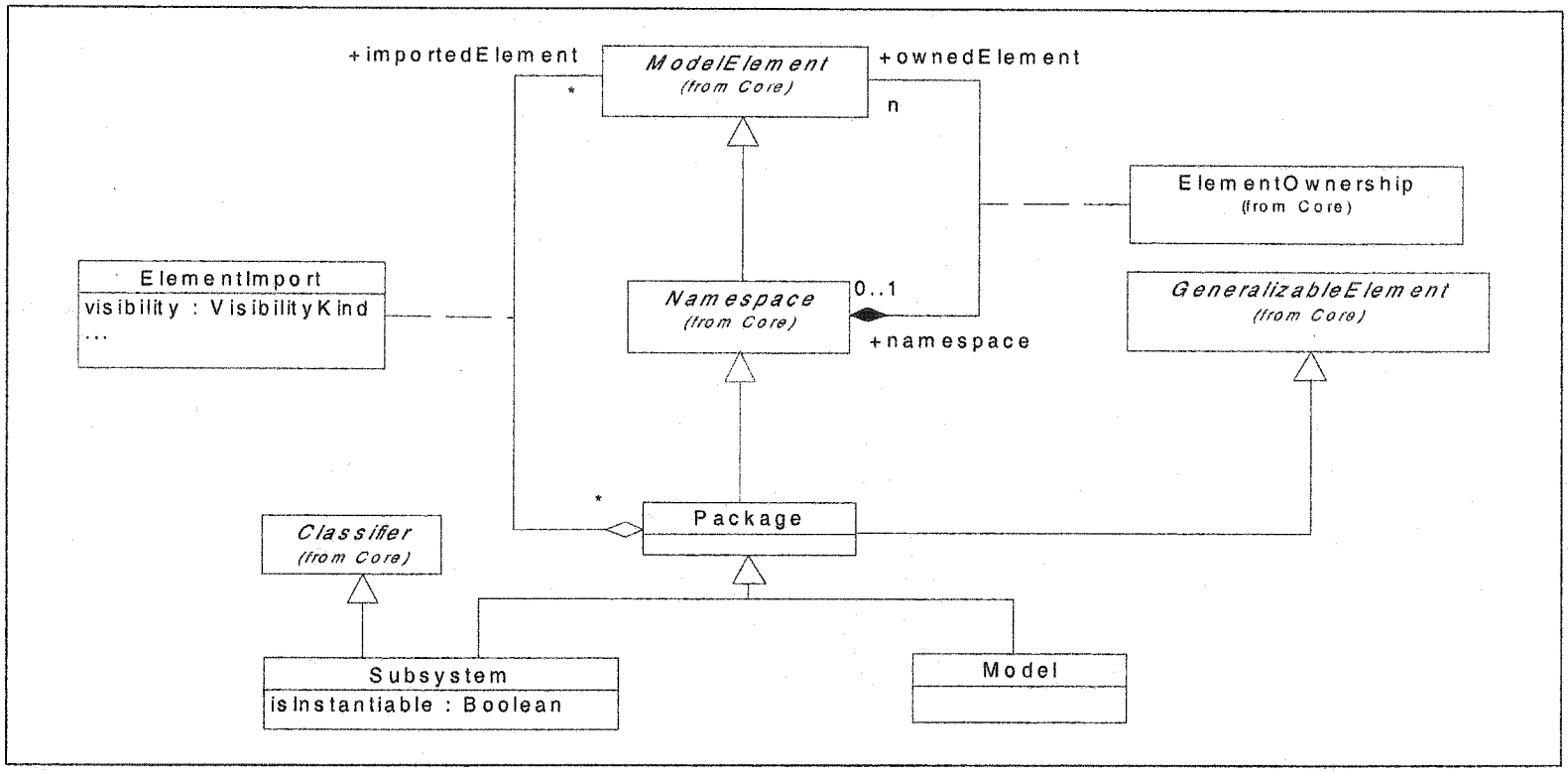

Figure 3:Model Management Package

The foundation package is the main package that defines different elements. It is divided into three packages; Core, Extension Mechanisms and Data Types as shown in its package diagram of Figure 4.

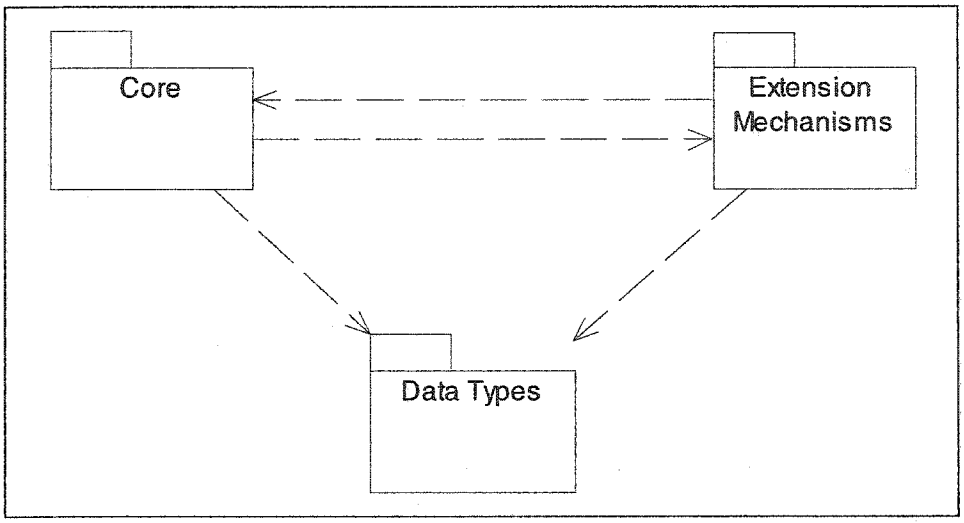

Figure 4: UML Foundation Package

The Core package is the most fundamental sub package of UML Foundation package. It defines the basic abstract and concrete metamodel constructs needed for the development of object models. Abstract constructs defined in the Core include ModelElement, GeneralizableElement, and Classifier. Concrete constructs specified in the Core include Class, Attribute, Operation and Association as shown in Figure 5 with attributes and operations on each entity suppressed.

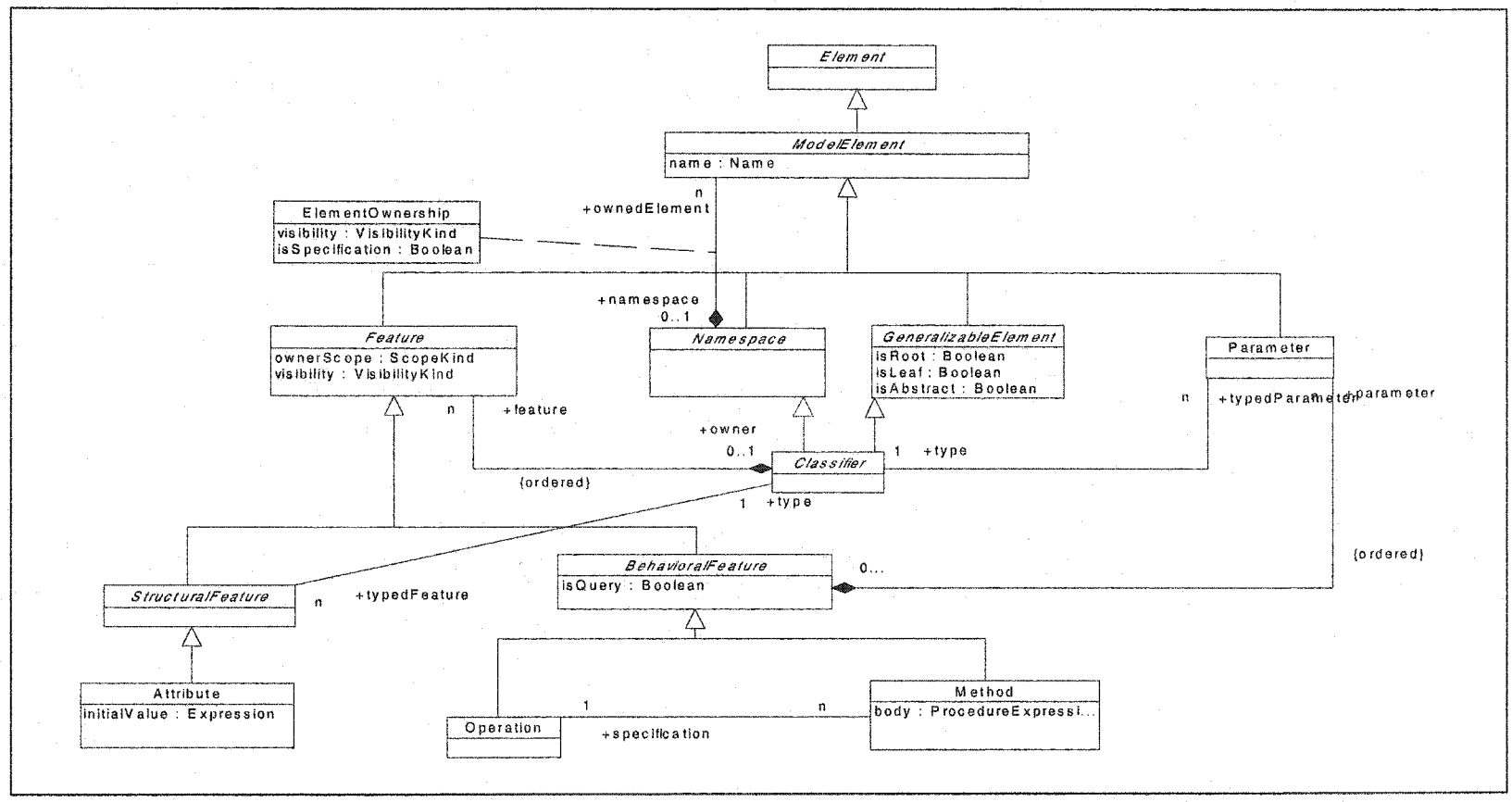

Figure 5: The Core UML package

The next important package for this work is behavioral package that contains the entities required for specifying the behavioral features of a system, as shown in Figure 6.

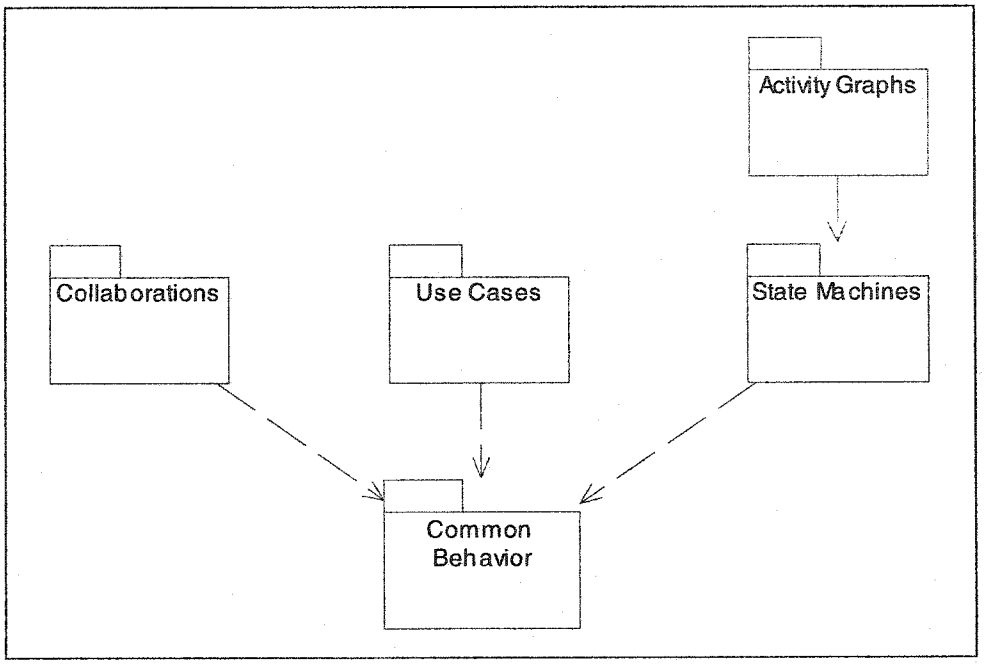

Figure 6: UML Behavioral Package

This package contains the sub-package Collaborations that contains the elements for specifying the sequence diagrams as shown in Figure 7. It is one of the packages that will be used in XMI Processing (6.5) for deriving the coupling.

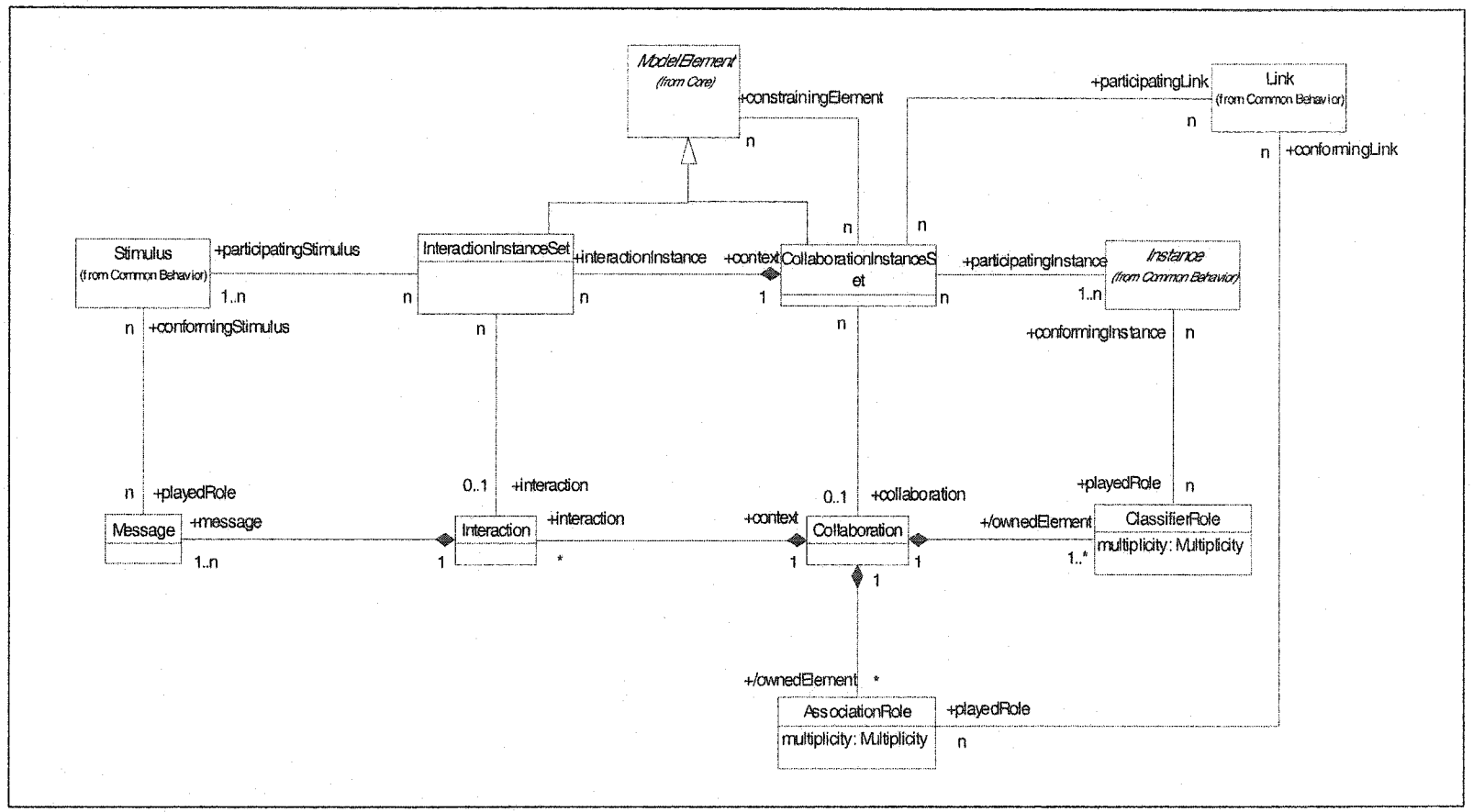

Figure 7: UML Collaboration Package

2.5.2.2 UML Notations

UML has different diagrams for visual depiction of different artifacts in different views. Using the above packages UML defines nine major kinds of modeling diagrams. We describe below each of the diagrams from its notation [18].

2.5.2.3 Package diagrams

A package is a collection of logically related UML elements. In Object oriented terms it represents components that are collection of classes.

2.5.2.4 Class diagrams

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static, they display what interacts but not what happens when they do interact. A Class is divided into two main parts, attributes and method along with their parameters. Association (Generalization, Aggregation and Simple Association) between classes is also shown in the class diagram. Object diagrams show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships. Figure 8 shows a simple example of class diagram that we will use throughout this chapter.

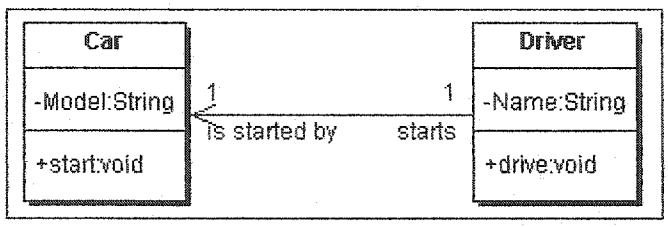

Figure 8: UML Class diagram example

2.5.2.5 Sequence diagrams

Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate. A sequence diagram is an interaction diagram that details how operations are carried out, what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence. A Message is a specification of Stimulus; i.e., it specifies the roles that the sender and the receiver Instances should conform to, as well as the Procedure that will, when executed, dispatch a Stimulus that conforms to the Message. The predecessor is a comma-separated list of sequence numbers followed by a slash ('/'): sequence-number ',' . . . '/'

The clause is omitted if the list is empty. Each sequence-number is a sequence-expression without any recurrence terms. It must match the sequence number of another Message. The meaning is that the Message is not enabled until all of the communications whose sequence numbers appear

in the list has occurred. Therefore, the list of predecessors represents a synchronization of threads. Note that the Message corresponding to the numerically preceding sequence number is an implicit predecessor and need not be explicitly listed. All of the sequence numbers with the same prefix form a sequence. The numerical predecessor is the one in which the final term is one less. That is, number 3.1.4.5 is the predecessor of 3.1.4.6. Sequence diagrams have two dimensions: 1) the vertical dimension represents time and 2) the horizontal dimension represents different instances or roles. Messages in sequence diagrams are ordered according to a time axis. This time axis is usually not rendered on diagrams but it goes according to the vertical dimension from top to bottom. Sequence diagrams do not use sequence numbers like collaboration diagrams to represent the message ordering. Message ordering is performed by the time axis.

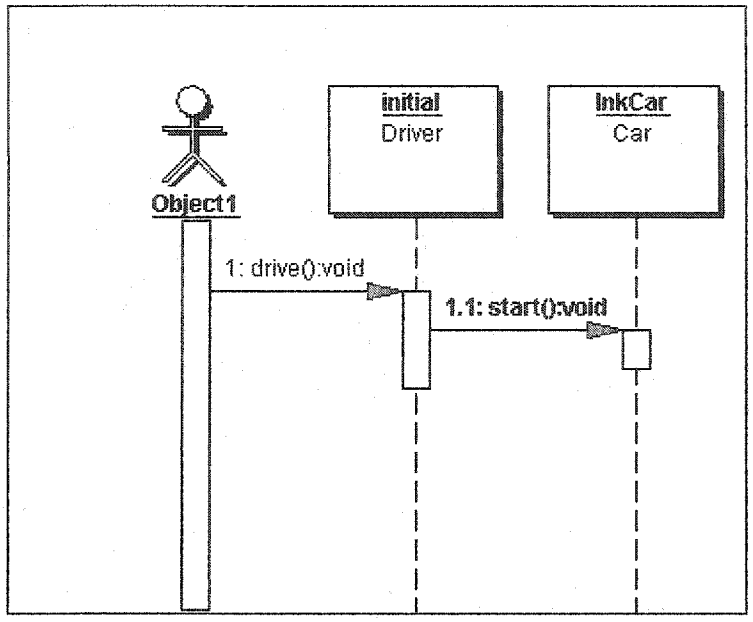

Figure 9: UML Sequence Diagram simple example

2.5.2.6 Collaboration diagrams

Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent, where as in a sequence diagram object roles are the vertices and messages are the connecting links. One can be generated from other and some case tools like Together [93] provide this functionality. Figure 9 shows the previous sequence diagram as collaboration diagram.

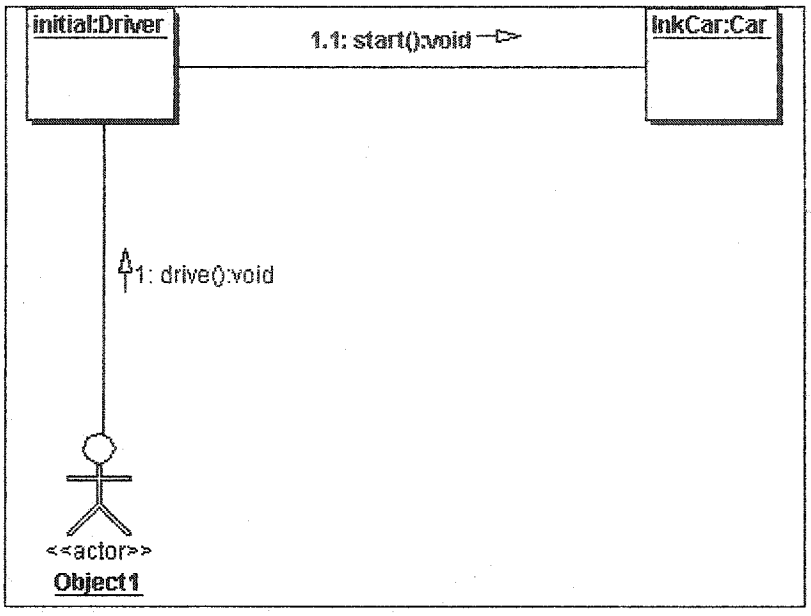

Figure 10: UML Collaboration diagram for Simple example

2.5.2.7 Use case diagrams

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how. Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system.

2.5.2.8 Statechart diagrams

Objects have behaviors and state. The state of an object depends on its current activity or condition. A statechart diagram shows the possible states of the object and the transitions that cause a change in state.

2.5.2.9 Activity diagrams

An activity diagram is essentially a flowchart with some additional features. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

2.5.2.10 Component diagrams

A component is a code module. Component diagrams are physical analogs of class diagram.

2.5.2.11 Deployment diagrams

Deployment diagrams show the physical configurations of software and hardware.

2.5.3 XMI

The XML Metadata Interchange Format (XMI) [20] is an XML based exchange format between UML tools, posted in response to an OMG Request for a Stream-based Model Interchange format. The main purpose of XMI is to enable easy interchange of data and metadata between UML

modeling tools and between tools and metadata repositories in distributed heterogeneous environments. XMI integrates three key industry standards:

- XML - eXtensible Markup Language, a W3C standard [19] that provides the universal format for structured documents and data on the Web.
- UML - Unified Modeling Language, the OMG modeling standard for specification, visualization, construction, and documentation of object-oriented systems;
- MOF - Meta Object Facility, a CORBA-compliant architecture for defining and sharing semantically rich metadata in distributed a heterogeneous environment that is used as OMG modeling and metadata repository standard.

XMI can be used to store not only UML model but also any MOF compliant model. XMI specification is very detailed and contains many features like document modification, validation, merging etc, that are outside the scope of this work.

Chapter 3

Coupling Taxonomy

In this chapter we discuss different frameworks according to which coupling has been classified. Most comprehensive among them is Briand's framework and it has been discussed in detail. We extend Briand's framework to include the concepts that they did not consider like interfaces and dynamic coupling.

3.1 Classification of Coupling

Berard [34] classified coupling into two types Necessary Coupling and Unnecessary Coupling. Necessary coupling was the coupling inherent in the problem domain and that is necessary to support the interactions among the program modules. Rest of the coupling is unnecessary. It is unnecessary coupling that signifies bad design and introduces many problems in the program. Hence it should be minimized.

Wild [42] classified coupling into Interface coupling and Internal Coupling. Interface coupling occurs when one object refers to one or more items in the public interface of another object. He classified Internal coupling as Inside Internal and Outside Internal coupling. Inside Internal is the coupling that

occurs among the components of an object or class. The outside internal coupling occurs between two classes through members other than those present in public interface.

Basili et. al. [38] classified coupling into three major types of Interaction, Component and Inheritance Coupling. Briand et al.[46] proposed a suite of metrics for C++ and extended Basili et. al. [38] classification. Inheritance, Interaction and Component coupling were identified as three dimensions of coupling. Some of the factors affecting these were: strength, locus and relationship type of the coupling connection, stability of server class and direct or indirect coupling. Strength of coupling depends on the frequency of connections between the classes and the type of connections. Locus refers to the direction of coupling. A coupling is of type export if the class is using the service or is at the client side of connection. The coupling of class that is at the other end of connection, providing a service or acting as server is of type import. Relationships refer to inheritance, peers etc. Type refers to the type of connection, like method-to-method, attribute-to-method, etc. Export coupling measures were found to be related to fault-proneness. The frequency with which a class is being used by other classes may not be indicative of the cognitive complexity of a class: a class could be used by many other classes, but still be constructed in a simple fashion. However, the modification of a class with high export coupling is critical, because it may

require follow-up modifications that potentially impact large parts of the system. Stability of server class cannot be measured automatically. Direct connections are those connections that occur at the first level. For example if a class A is connected with class B through two connections and class B is connected through three connections to class C, then direct connections of class A are only two. Indirect Coupling measurement considers apart from connections at first level, other connections that may be connected after the first level. Thus the definition is of recursive nature. In previous example the indirect connections of class A will be five (two direct at first level and three at second level). Hence if connection is defined as a relation then Indirect Coupling considers the transitive closure of this relation. RFC (Response Set of a method) metric can be computed to account for number of methods that can be invoked from a method.

Briand et. al. framework [46] is the most comprehensive work that is done to collect all the coupling metrics and classify them. They introduced a formalism to represent all the concepts that contribute towards coupling. They considered three main coupling metric suites [38][43][44] and then classified them according to this new framework. The classification of coupling is significant in reducing the ambiguity and differentiating between so many coupling metrics that were proposed. Table 3 presents Briand et. al. formalism in some detail. We extend this formalism to include some more

concepts like interfaces. Table 3 summarizes the classification of coupling metrics and factors affecting each coupling type.

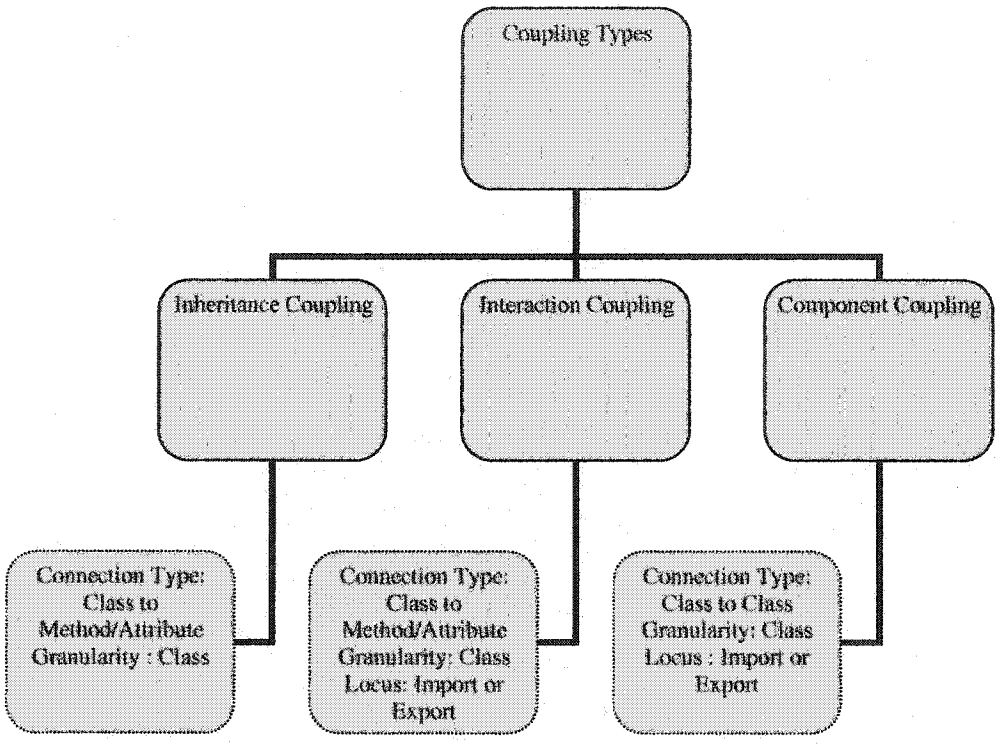

Figure 11: Briand's Coupling Taxonomy

Table 3: Briand's Coupling Formalism

Concept	Explanation	Formalism
Inheritance	Sets of parent, children, ancestor, and descendent classes of a class c where C represents the set of all classes.	$Parents(c) \subset C$ $Children(c) \subset C$ $Ancestors(c) \subset C$

Concept	Explanation	Formalism
		$Descendents(c) \subset C$
Methods	Set of all methods in the system, M(C) can be found from union of set of methods each class c, M(c)	$M(C) = \bigcup_{c \in C} M(c)$
	The first set in first equation denote the set of methods declared in c and the first set in second equation shows the set of methods implemented in c	$M_D(c) \subseteq M(c)$ $M_I(c) \subseteq M(c)$ $M(c) = M_D(c) \cup M_I(c)$ $M_D(c) \cap M_I(c) = \Phi$
	The first set in first equation denotes the set of inherited methods of c. The first set in second equation shows the overriding methods of c. The first set in third equation denotes the set of new methods of c.	$M_{INH}(c) \subseteq M(c)$ $M_{OVR}(c) \subseteq M(c)$ $M_{NEW}(c) \subseteq M(c)$ $M_{INH}(c) \cup M_{OVR}(c) \cup M$

Concept	**Explanation**	**Formalism**
Method Invocations from each method $c \in C$ $m \in M_I(c)$ $m' \in M(c)$	Set of statically invoked methods SIM and their number NSI.	$m' \in SIM(m) \Leftrightarrow \exists d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' is invoked for an object of status type class d and $NSI(m,m')$ denotes number of such static invocations of m' by m.

Concept	Explanation	Formalism
	Set of polymorphically invoked methods PIM and their number NPI. Thus one method invocation can contribute to the NPI count of several methods, because of polymorphism.	$m' \in PIM(m) \Leftrightarrow \exists d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' may, because of polymorphism, be invoked for an object of dynamic type class d and $NPI(m,m')$ denotes number of such dynamic invocations of m' by m.
Attributes	Set of attributes declared in class c	$A_D(c)$
	Set of attributes implemented in class c	$A_I(c)$
	Set of all attributes of class c	$A(c) = A_D(c) \cup A_I(c)$

Concept	Explanation	Formalism
	Set of all attributes of system	$A(c) = \bigcup_{c \in C} A(c)$
Attribute References	Set of all attributes referenced by the method m.	$AR(m)$
Types	Basic built-in-types by language User defined types of global scope All available types in system Type of attribute or type of parameter of method m.	BT UDT $T = BT \cup UDT \cup C$ $T(a) \in T$ $T(v) \in T$ *where* $v \in Par(m)$
Predicate uses	A class c uses a class d if a method implemented in class c references a method or an attribute implemented in class d.	$uses(c,d) \Leftrightarrow$ ($\exists m \in M_I(c):$ $\exists m' \in M_I(d):$ $m' \in PIM(m)$) ∨ ($\exists m \in M_I(c)$ $\exists a \in A_I(d):$ $a \in AR(m)$)

This formalism can be used to express any object oriented coupling metric without any ambiguity. For example CBO [37] metric can be defined formally as

$$CBO(c) = \left| \{ d \in C - \{c\} \} \middle| uses(c,d) \vee uses(d,c) \} \right|$$

3.2 Ambiguity in coupling metrics

The number of coupling metrics proposed in literature has grown in large numbers. Among the large numbers of coupling measures the choice of the measure is difficult. Reasons for different understanding of coupling metrics are

- Different understanding of core concept
- Difference in objective of measuring coupling
- Difference in the underlying programming paradigm

In this section we will discuss some limitations of existing taxonomy for coupling and propose some enhancements to it.

3.3 Issues with present taxonomy

In this section we present some of the concepts that earlier taxonomies left out, and that now have become important such as interfaces and dynamic metrics.

3.3.1 Polymorphic Invocation

If a method m invokes another method m' of class d then either it is static invocation or dynamic invocation. The dynamic invocations because of polymorphism contribute to NPI count of several methods. But Briand et. al. ignored the fact that whether in case of overriding method, the method calls its parent method also. If it calls parent method then it implies it is coupled to both parent as well as child even if it is a case of static invocation. Hence this implies that even static invocation can contribute to NSI count of several methods that are parents of the method's class. Hence we introduce a new formalism for taking into account this difference. This is significant for it shows that a child class' method is coupled to its parent's counterpart and can have an effect on computing inheritance metric.

Summarizing PIM can take care of method invocation traveling down the hierarchy whereas the new formalism can take care of method invocation going up the hierarchy.

3.3.2 Granularity

In some of the metrics proposed, it has been observed that the module level measures are aggregated and shown as total coupling of that module. We see this more of cohesion than coupling. Similarly 'internal coupling' or 'inside internal coupling' proposed by Berard et.al. [34] that measures how much a method uses its own members and friends classes in case of C++. Coupling is always between modules, (modules may be classes, packages, systems etc), and whenever we have a metric for single module then it has to be analyzed whether it is cohesion or how it can be better computed to express cohesion. We do calculate total coupling for classes and packages but that only shows total import coupling of that class with all others.

3.3.3 Nature of coupling

Recent studies in coupling have identified the importance of dynamic coupling (2.3), hence it is one of the dimensions of coupling. But it does not apply for all connection types, for example for inheritance connection it does not make any difference, but for interaction connections static and dynamic values may differ.

3.3.4 Strength of coupling

Most of the coupling metrics and Briand et. al. classification does not take into account the module size when considering strength or magnitude of

coupling. But size of module affects coupling. The bigger the module that is at another end of connection the stronger should be the coupling.

3.4 Extension to coupling formalism

The Table 4 extends the earlier Briand's et. al. [46] formalism presented in Table 3 with the new concepts discussed above.

Table 4: Extension to coupling formalism

Concept	**Explanation**	**Formalism**
Method Invocations from each method $c \in C$ $m \in M_I(c)$ $m' \in M(c)$	Set of statically invoked methods SIM and their number NSI.	$m' \in SIM(m) \Leftrightarrow \exists d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' is invoked for an object of status type class d and $NSI(m,m')$ denotes number of such static invocations of m' by m.

Concept	Explanation	Formalism
	Methods invoked are represented by messages in UML. Thus message can be introduced as some metrics use messages to define coupling	message(m,C_{send} C_{rec})
	Set of polymorphically invoked methods PIM and their number NPI. Thus one method invocation can contribute to the NPI count of several methods, because of polymorphism.	$m' \in PIM(m) \Leftrightarrow \exists d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' may, because of polymorphism, be invoked for an object of dynamic type class d and $NPI(m,m')$ denotes number of such dynamic invocations of m' by m.
Interfaces	Set of all interfaces in system I	$I(c) \subseteq I$

Concept	Explanation	Formalism
	Set of interfaces implemented by a class	

This framework can be used to express new coupling metric that can take into account the effect of interface, association, and composition on coupling. Figure 12 summarizes the classification of coupling metrics into different types and the factors affecting each type.

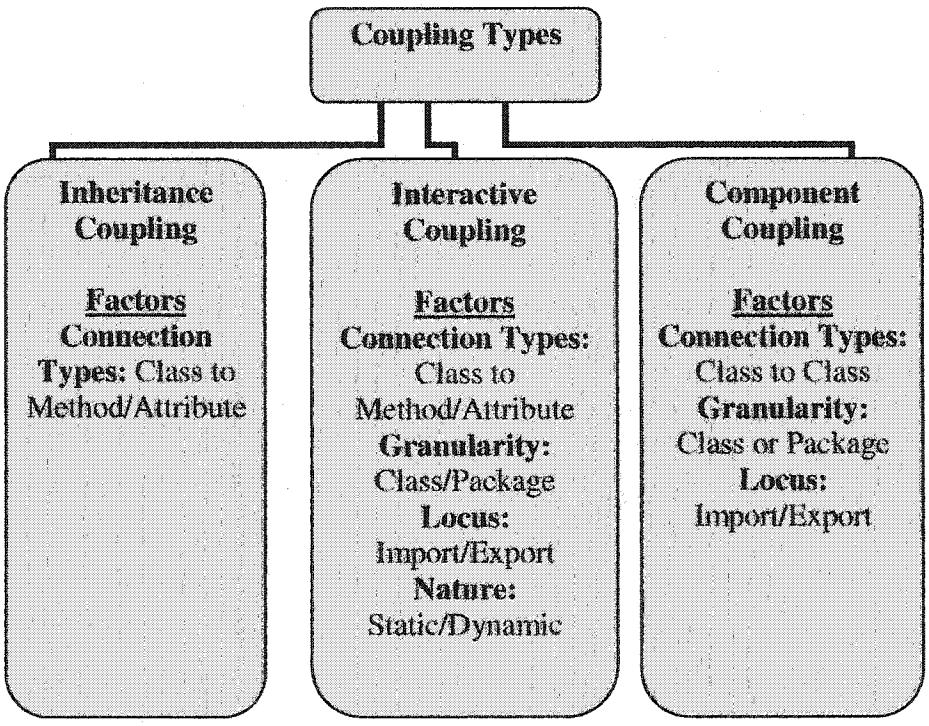

Figure 12: Coupling Taxonomy

Chapter 4

Metrics at Design Level and Source Level

This chapter discusses the motivation for design level metrics, how do they differ from source level metrics and a comprehensive approach for computing design level metrics from UML and XMI.

4.1 Motivation for Design Level Metrics

Design Level Metrics offer the following advantages over source level Metrics:

- An insight into the software architecture
- Early design critic information
- Flaws in the design
- Predicting most critical parts of the design

4.2 Design Level Vs Source Level Metrics

Design level metrics are those metrics that can be computed from software design. Source level metrics are those metrics that can be computed from source code. This leads to an important question "what is design"? Subrata

[15] classifies the concept into two schools of thought: one that views it as a blueprint for implementation and the other view of formalists who view programs as mathematical objects and design as an exercise in applied mathematics. Whichever view we take it is clear that source contains design. However, source code may include details that are not part of the design. Hence any design metric can be computed from the source code but not vice versa. Whether a source metric can be computed from the design depends on how the design is expressed and how does design leads to source. This brings software implementation process into picture. For expression of design we consider only UML as it is the de facto language for design now. Water fall model consists of fixed number of stages of development like analysis, design, development and testing that strictly follow one another. In such a model the whole design always precedes code. Hence code may reveal a different design than the design that was freezed in the design phase. Hence the difference between design and source metric will reveal the change that took place between design phase to implementation phase. If the design is reverse engineered from source and the difference between design and source metric is how much UML design differ with explicit design in source. In RUP (Rational Unified Process) the waterfall model is followed in iterations. All activities like analysis, design, implementation and testing are repeated in every iteration. The difference between design and source metric

will be less than Waterfall model and represent the difference between two iterations. In Agile process the iterations are still smaller and round trip engineering is one of the major features in agile methodology. Hence source and design are always in synchronization. In such a case the difference between design and source metric will reveal how much UML design differ with explicit design in source.

4.3 Coupling at Design and Source level

As discussed earlier most of metrics proposed and implemented in literature have been extracted from the source code. This applies more for coupling metrics. Hence it is important to define coupling in more detail in the context of whether it is source level property or design level. Before we present our view let us see what other researchers have to say on this.

Trevor Paterson [78] mentions, "in the absence of code implementation it is not possible to derive any of Chidamber and Kemerer's metrics pertaining to cohesion and coupling from UML diagrams"

Reibing (2001) adds that although it is theoretically possible for UML diagrams to include relations between operations and attributes, information that could be used to calculate these metrics, this level of detail would rarely be included in a UML diagram, particularly at an early design stage.

Scott A. Whitmire [76] makes an interesting and contrary claim to the above view, "Amount of coupling between modules is completely dependent on the design, if the design was taken to that level." He also notes that measuring coupling from source code is a form of indirect measurement.

Briand et. al.[46] states that there is no coupling measure that can be classified as usable at the analysis or HLD (High Level Design) phase. Some are partially usable, but empirical studies are required to analyze how accurate such approximations are and whether they are useful predictors of external attributes.

Finally we conclude that coupling is a design level as well as source level property. Hence its computation from design and source will differ depending on the language used to express design. In case of UML following points can be concluded:

- Inheritance Metrics if based on usability information (whether the sub-class is using the parent class' methods or attributes), like Usability Inheritance Metric [27] it cannot be accurately computed from the design. On the other hand Inheritance Metrics like AlMulla's [28] metric that is based on visibility of class members can be accurately computed from the design. Component Coupling can be

accurately computed from design if it is based on associations between classes.

- Since Interaction coupling is based on interactions among the classes that are only partially captured in sequence diagram, the source will always have more complete information on the interactions. Thus Interaction Coupling cannot be accurately computed from the design because its computation from source will always differ from the design.

4.4 Conceptual framework for UML Metrics

This section gives a complete description of coupling measures that can be computed from UML design. We use the same formalism introduced earlier in Table 4. For each concept of formalism we see whether it can be measured from UML designs stored as XMI files. This is summarized in Table 5. For each concept we present the formalism, the UML diagram needed to measure it and the XMI nodes needed to extract the information required for measurement.

Table 5: Coupling Measures possible with UML

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
Inheritance	Sets of parent, children, ancestor, descendent classes of a class c where C represents the set of all classes.	$Parents(c) \subset C$ $Children(c) \subset C$ $Ancestors(c) \subset C$ $Descendents(c) \subset C$	Class Diagrams	Foundation.Core.GeneralizableElement.generalization Foundation.Core.GeneralizableElement.isRoot Foundation.Core.GeneralizableElement.isLeaf
Methods	Set of all methods in the system, M(C) can be found from union of set of methods each class c, M(c)	$M(C) = \bigcup_{c \in C} M(c)$	Class Diagrams	Foundation.Core.Operation
	The first set in first equation denote the set of methods declared in c and the first set in second equation shows the set of methods implemented in c	$M_D(c) \subseteq M(c)$ $M_I(c) \subseteq M(c)$ $M(c) = M_D(c) \cup M_I$ $M_D(c) \cap M_I(c) = \Phi$	Class Diagrams	Foundation.Core.Operation and Foundation.Core.GeneralizableElement.generalization

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
	The first set in first equation denotes the set of inherited methods of c. The first set in second equation shows the complete overriding methods of c. The first set in third equation shows the overriding methods of c. The first set in fourth equation denotes the set of new methods of c.	$M_{INH}(c) \subseteq M(c)$ $M_{COVR}(c) \subseteq M(c)$ $M_{OVR}(c) \subseteq M(c)$ $M_{NEW}(c) \subseteq M(c)$ $M_{INH}(c) \cup M_{OVR}(c)$	Class Diagrams	Foundation.Core.Operation and Foundation.Core.GeneralizableElement.generalization Not possible Not possible Comparing the above two nodes in child and parent

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
Method Invocations from each method $c \in C$ $m \in M_I(c)$ $m' \in M(c)$	Set of statically invoked methods SIM and their number NSI.	$m' \in SIM(m) \Leftrightarrow \exists d$ such that $m' \in M(d)$ and the body of m has a method invocation where m' is invoked for an object of status type class d and $NSI(m,m')$ denotes number of such static invocations of m' by m.	Sequence Diagrams and Class Diagrams	Behavioral_Elements.Collaborations Behavioral_Elements.Collaborations.Interaction Behavioral_Elements.Collaborations.ClassifierRole Behavioral_Elements.Collaborations.Message Behavioral_Elements.Collaborations.Message.sender Behavioral_Elements.Collaborations.Message.receiver Behavioral_Elements.Common_Behavior.CallAction

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
	Methods invoked are represented by messages in UML. Thus message can be introduced as some metrics use messages to define coupling	message(m,C_{send} C_{rec})	Sequence Diagrams	Behavioral_Elements.Collaborations.Message Behavioral_Elements.Collaborations.Message.sender Behavioral_Elements.Collaborations.Message.receiver Behavioral_Elements.Common_Behavior.CallAction

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
	Set of polymorphically invoked methods PIM and their number NPI. Thus one method invocation can contribute to the NPI count of several methods, because of polymorphism.	$m' \in PIM(m) \Leftrightarrow \exists d$ such that $m' \in M(d)$ and the body of m has a method invocation where m' may, because of polymorphism, be invoked for an object of dynamic type class d and $NPI(m,m')$ denotes number of such dynamic invocations of m' by m.	Sequence Diagrams and Class Diagrams	Behavioral_Elements.Collaborations Behavioral_Elements.Collaborations.Interaction Behavioral_Elements.Collaborations.ClassifierRole Behavioral_Elements.Collaborations.Message Behavioral_Elements.Collaborations.Message.sender Behavioral_Elements.Collaborations.Message.receiver Behavioral_Elements.Common_Behavior.CallAction
Attributes	Set of attributes declared in class c	$A_D(c)$	Class Diagrams	Foundation.Core.Attribute
	Set of attributes implemented in class c	$A_I(c)$		Not possible
	Set of all attributes of class c	$A(c) = A_D(c) \cup A_I(c$	Class Diagrams	Foundation.Core.Attribute

Concept	**Explanation**	**Formalism**	**UML diagrams required**	**Relevant XMI (1.1) Node Types (only the parent nodes)**
	Set of all attributes of system	$A(c) = \bigcup_{c \in C} A(c)$	Class Diagrams	Foundation.Core.Attribute
Attribute References	Set of all attributes referenced by the method m.	$AR(m)$	Sequence Diagrams and Class Diagrams	(Partially) If the attributes are of the type objects and also used to invoke some method
Types	Basic built-in-types by language User defined types of global scope All available types in system Type of attribute or type of parameter of method m.	BT UDT $T = BT \cup UDT \cup C$ $T(a) \in T$ $T(v) \in T$ *where* $v \in Par(m)$	Class Diagrams	(Partially) Foundation.Data_Types It may not be possible to differentiate between built in data types and system global types. Foundation.Core.Parameter.type

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
Predicate uses	A class c uses a class d if a method implemented in class c references a method or an attribute implemented in class d.	$uses(c,d) \Leftrightarrow$ ($\exists m \in M_I(c):$ $\exists m' \in M_I(d):$ $m' \in PIM(m)$) $\vee$ ($\exists m \in M_I(c)$ $\exists a \in A_I(d):$ $a \in AR(m)$)	Sequence Diagrams and Class Diagrams	Behavioral_Elements.Collaborations Behavioral_Elements.Collaborations.Interaction Behavioral_Elements.Collaborations.ClassifierRole Behavioral_Elements.Collaborations.Message Behavioral_Elements.Collaborations.Message.sender Behavioral_Elements.Collaborations.Message.receiver Behavioral_Elements.Common_Behavior.CallAction (Partially)
Interfaces	Set of all interfaces in system I Set of interfaces	$I(c) \subseteq I$		Foundation.Core.Operation and Foundation.Core.Generalizabl

Concept	Explanation	Formalism	UML diagrams required	Relevant XMI (1.1) Node Types (only the parent nodes)
	implemented by a class			eElement.generalization
Association	Associations between classes	A(c,d)	Class Diagrams	Foundation.Core.Association Foundation.Core.Association End

In most cases to measure a certain concept, information about the child nodes is needed. The child nodes are not mentioned in Table 5. For example to know all the classes of system, we need class diagrams. Now in the XMI file of this class diagram we need all "Foundation.Core.Class" nodes. For details of each class the children of this node can be traversed.

Table 5 can be used to classify all coupling measures that can be computed from UML as well as the required UML diagrams to compute those metrics.

In Table 6 we express the coupling metrics possible with UML in terms of the entities required in UML. It also indicates the cases where the computation of metric from source and design will be different and the cause of the difference.

Table 6: Metric Dependency Table

Coupling Type and nature	Coupling Metric	Elementary Metrics Required	Metric Definition	UML Designs Required	Source Vs Design
Inheritance Static	Inheritance Coupling (IC)	DIT, NOC	DIT (depth of inheritance tree) + NOC (number of children)	Class Diagram	Same
Interaction Static	Class Coupling (CBO)		The number of access to other classes + The number of access by other classes +The number of co-operated classes. (A co-operated class is a class in which a method requests some services from another class and vice versa.)	Class Diagram and Sequence Diagram	Different because of details like method invocations
Inheritance Static	Potential Inheritance Coupling	Class size	Definition Matrix and Coupling Matrix	Class Diagram	Different because of class size
Interaction Static	UML Interaction Coupling	UML Class size	Number of method uses weighed by the class size	Class Diagram and Sequence Diagram	Different because of class size
Component	UML Component Coupling	UML Class size	Number of associations weighed by the class	Class Diagram	Different because of class size

Coupling Type and nature	Coupling Metric	Elementary Metrics Required	Metric Definition	UML Designs Required	Source Vs Design
			size		
Interaction Static	Package Coupling	UML Interaction Coupling	Interclass Coupling matrix	Package Diagram, Class Diagram and Sequence Diagram	Different
Interaction Dynamic	Dynamic UML Interaction Coupling		Same as UML Interaction coupling except dynamic invocations will be used	Package Diagram, Class Diagram and dynamic Sequence Diagram	Different
Inheritance Dynamic	Dynamic UML Inheritance Coupling		Same as UML Inheritance coupling except dynamic invocations will be used	Package Diagram, Class Diagram and dynamic Sequence Diagram	Different

None of dynamic coupling measures proposed in literature tried to use dynamic sequence diagrams for dynamic coupling. Dynamic sequence diagrams can be generated from the traces of running application. Dynamic coupling has been claimed to give insight into several quality attributes of system like fault tolerance [59][65][66][68]. Therefore we introduce it as another dimension of coupling. Sequence diagrams can be used for dynamic coupling, if they are derived from the execution trace. The only tool we came across is J2U [63] that instruments byte code of a java program and generate the sequence diagrams from the trace of running program. But the tool is not mature enough to be used for large size programs.

Chapter 5

UML Coupling Metrics

Most coupling metrics including proposed coupling metrics use class size as an elementary metric for computation of coupling. Hence the proposed metric for UML class size is discussed in section 5.1 as an elementary metric for computing other higher level metrics. We did not use the existing metrics for UML Class size like number of attributes, methods because they are very rough estimates of class size at design level as well as code level. Section 5.2, 5.3, 5.4 and 5.5 discuss the Inheritance coupling metric, component coupling metric, interactive coupling metric and package coupling metric for UML, respectively.

5.1 UML Class Size

The design artifacts that are available in early design phase are Use Cases, Package Diagrams, Class Diagrams and Sequence Diagrams. Class diagrams and Sequence diagrams are used to compute the Class Size metric because class and sequence diagrams map directly to source code. Hence the metric could be used as better predictor of actual size of source code. We used the following weights for complexity of a data type, they are proportional to

their size in Java, and are mostly proportional across other programming languages as well.

Table 7: Attribute Size

Attribute Type	**Attribute Size**	**Attribute size in Java (in bytes)**
Int	4	4
Byte	1	1
Short	2	2
Long	8	8
Float	4	4
Double	8	8
Char	2	2
Boolean	1	1
String or any other object type	20	Variable

A similar attribute weighing scheme was proposed in [29], but the weights and attribute types were based on *C* language and some data types existing in java were not there.

Method size in UML can be estimated from the size of each of parameters and the return value and messages exchanged by the method. The more the numbers of messages, it will increase the size of method. For method m, the method size can be computed as:

$$Size(m) = (\sum_{i=1}^{np} Size(param_i) + Size(return)) \times n(message)$$

where:

- np is the number of parameters in method m

It should be noted above that the messages considered are only the messages that travel between classes. The messages within the class are ignored because the information is only available later in the development lifecycle.

Finally the UML Class Size can now be defined as the sum of each of its constituents like attribute, method and inner classes. The class size is estimated from the size of each of its attributes, methods and the size of its inner classes as follows:

$$Size(c) = \sum_{i=1}^{na} Size(att_i) + \sum_{i=1}^{nm} Size(meth_i) + \sum_{i=1}^{ni} Size(innerClasses)$$

where:

- na is the number of attributes
- nm is the number of methods
- ni is the number of inner classes

5.2 UML Inheritance Coupling

UML inheritance coupling (UINC) metric is based on the inheritance coupling metric proposed by AlMulla [28]. For class size UML class size metric was used. Detailed information like Polymorphic method calls and used attributes are not available in UML design artifacts as shown in Table 5. Therefore most of the inheritance coupling metrics give less information about use of inheritance. AlMulla's metric does not depend on details. It estimates the inheritance coupling from the visibility of class members. Therefore it is the most appropriate metric for computing inheritance coupling at design level.

AlMulla's coupling metric use a measurement framework proposed by AlGhamdi [26]. This framework was also used by Elish [27] in computing Usability Inheritance Coupling metric.

5.2.1 Measurement Framework

In AlGhamdi et. al. [26] measurement framework there is a definition matrix consisting of classes represented as rows and attributes and methods represented as columns. Each entry represents the strength or weight of connection between a class and the respective attribute or method. The entries of definition metric are computed using the following formula:

$$d_{ij} = w_i + \sum_{k=0}^{d-1} (NOC_k * (d - k))$$

where:

- w_i is the attribute or method complexity
- d is the longest length from current class to any of its descendents
- NOC_k is the number of descendents of current class that inherit the attribute or method.

Table 8: Definition Matrix

	Attribute$_1$	..	**Attribute$_{an}$**	..	**Method$_1$**	...	**Method$_{mn}$**
Class$_1$							
.							
Class$_{cn}$							

where

- an is the number of attributes
- mn is the number of methods
- cn is the number of classes

From this definition matrix the coupling matrix is derived using following formula:

$$C_{ij} = \sum_{k=1}^{n} \alpha_i * D_{jk} * \beta_k * D_{jk}$$

where:

- α_i is the inverse of sum of i^{th} row
- β_k is the inverse of k^{th} column of definition matrix

Table 9: Coupling Matrix

	$Class_1$	.	.	**$Class_n$**
$Class_1$				
.				
.				
$Class_n$				

Al-Mulla [28]used this framework for computing his metrics, and he considered weights for every attribute or method that can be visible in any descendents. Hence we can say that AlMulla's coupling metric measures potential coupling. This potential coupling should also be the maximum static coupling that can exist.

Elish [27] used the same framework but he computed the weights in definition metric only if the sub classes really used the attribute or method, otherwise they were not considered. Hence he called his metric Usability Based Coupling Metric. But Elish's Metric can only be roughly estimated with UML because it is based on usability information that is only partially available in UML.

The following are the interpretations of coupling metric that were used in all of the above works:

- Each non-diagonal entry is the coupling between classes indicated by the column and the row. Thus C_{ij} is the coupling between classes C_j and C_i.
- Each diagonal entry shows the separation of the respective class from the rest of the system.

5.3 UML Interactive Coupling

UML Interactive Coupling (UIC) represents the interclass interactive coupling that exists between different classes of the system.

AlGhamdi framework [25] for computing inheritance coupling can not be used for interactive coupling, because of following issues:

- If we increase the weight of an attribute for another class, the self-coupling of the class changes. This leads to violation of one of the axiom [114] that is used to theoretically validate coupling metrics. This axiom states that merging of unconnected classes should not affect coupling at all.
- The framework does not take into account the direction of coupling, which is necessary for interactive coupling.

We propose a simpler framework for measuring interactive coupling that takes into account the weights for every type of connections between classes. In this case we can represent definition matrix as in Table 10 where each entry in the matrix represents the total weight of connection between the client class represented by row and server class represented by column. The types of connection included can be defined depending on the type of coupling metric being computed. In this metric we have included all the

connections that can be computed from UML that were summarized in Table 5. Table 5 shows that the predicate *uses* can be partially measured from UML. This predicate means that a class is using an attribute or a method of another class. It can be partially measured because the information about attribute use is not always available in design stage. Hence we consider only the method use for computing the interaction coupling. Hence the UML Interactive Coupling Metric gives a comprehensive measure of interactive coupling in the design.

$$d_{ij} = \sum_{k=0}^{nm} MethodSize(uses(i, j)) \quad where \quad i \neq j$$

$$d_{ij} = 0 \quad where \quad i = j$$

where:

- nm is the number of methods that class i is using from class j

Table 10: Definition Matrix and Coupling Matrix for UIC

	$Class_1$	.	**$Class_n$**	**Overall Class Coupling**
$Class_1$				$\sum_{j=1}^{n} d_{1j}$
.				
$Class_n$				$\sum_{j=1}^{n} d_{nj}$

For example in the Sequence diagram shown in Figure 13, there is only one interaction connection between the two classes shown. The first class is importly coupled with the second class and the weight of this connection is given by the method weight. The method weight as discussed before.

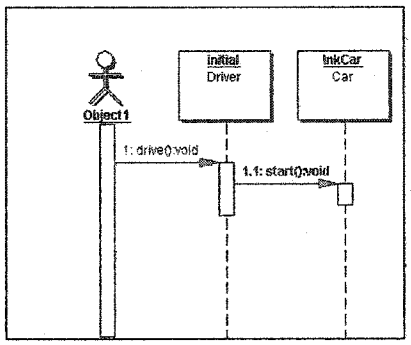

Figure 13: Interaction Connection

We sum all the entries of definition matrix and finally normalize all the entries with this sum to come up with a coupling matrix.

$$S = \sum_{i=0}^{n} \sum_{j=1}^{n} d_{ij}$$

$$C = D \div S$$

where

- n is the number of classes in system
- D is Definition matrix of size n * n

- C is Coupling matrix of size n * n
- S is scalar

Although a very simple framework for computing coupling matrix this framework offers following advantages:

- Each entry C_{ij} represents the import coupling between class C_i and class C_j.
- Diagonal entries are zero and have no effect on coupling calculations
- Since the measures are normalized over the system, they serve as useful indicator of coupling relative to the other classes of the system.
- The Export coupling matrix can be easily derived, as it is just transpose of this matrix.
- If needed a finely detailed coupling can be computed like coupling due to a particular connection type.

5.4 UML Component Coupling

The interclass UML Component Coupling (UCC) takes into account the coupling between classes when a class is used as a type in another class' member variable. In UML this corresponds with associations. We consider only the class member variables and not the temporary variables or method

parameters. The same framework is used for computation as used for Interactive coupling. The definition matrix entries are computed by the following equation:

$$d_{ij} = \sum_{k=1}^{n} ClassSize(association_k)$$

As the equation shows it does not only take into account the number of associations but also the size of class that is associated. The coupling metric is derived from the definition matrix in the same way as for Interactive coupling. The example below demonstrates how the component coupling is computed between two classes. Here Class 2 has two associations with class 1. Hence the definition matrix entry between class 1 and class 2 will contain a definition weight of 4+4=8.

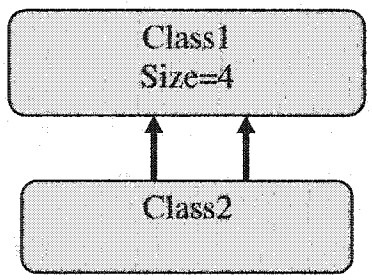

Figure 14: Component coupling example

5.5 Package Coupling

We use the interaction-coupling matrix (UIC Matrix) to compute the package coupling. We derive three measures of package coupling from the interaction-coupling matrix, the Inter-Package Coupling, Internal Package Coupling and External Package Coupling.

5.5.1 Inter-Package Coupling

Inter-Package Coupling (IPC) is the coupling that exists between different packages of the system. The UIC Matrix is traversed and for each entry of coupling matrix the packages of both the classes are checked and the coupling value is added to the respective cell in the Package Coupling Matrix. The following algorithm is used to compute the inter-package coupling.

Algorithm IPC

Input: UIC[nc][nc] (Inter-class Coupling matrix)

where nc is the number of classes.

Output: IPC[np][np] (Inter-package Coupling matrix)

where np is the number of packages in system.

1. **while** (i < nc) and (j < nc)

2. **if** (Package(C_i) < > Package(C_j)) **then**

3. IPC[Package(C_i)][Package(C_i)] + = UIC[i][j]

4. **end while**

Table 11: IPC (Inter-Package Coupling) Matrix

	Package$_1$	.	**Package$_n$**	
Package$_1$				$\sum_{j=1}^{np} IPC_{P1Pj}$
.				
Package$_n$				$\sum_{j=1}^{np} IPC_{PnPj}$

$$S = \sum_{i=0}^{np}\sum_{j=0}^{np} IPC_{PiPj}$$

where

- np is the number of packages in the system

The Package coupling matrix is normalized by the sum of all package couplings for easy comparison across packages.

5.5.2 Internal Package Coupling

Internal Package Coupling (INPC) represents the total coupling between the classes of same package. This measure may give an idea of the cohesion of

package itself. The UIC coupling matrix (Table 10) is traversed and for each entry of coupling matrix the packages of both the classes are checked and if both are same then the coupling value is added to the Internal Package coupling measure of that Package. The following algorithm is used to compute the inter-package coupling.

Algorithm INPC

Input: UIC[nc][nc] (Inter-class Coupling matrix)

where nc is the number of classes.

Output: INPC[np] (Internal Package Coupling matrix)

where np is the number of packages in system.

1. **while** (i < nc) and (j < nc)
2. **if** (Package(Ci) = = Package(Cj)) **then**
3. INPC[Package(Ci)] + = UIC[i][j]
4. **end while**

5.5.3 External Package Coupling

The External Package Coupling (EPC) represents the total coupling that a package has with all other packages of the system. From the Package

coupling matrix the sum of Package's coupling with external packages is summed as the External Package coupling measure of that Package.

$$EPC_{P_i} = \sum_{j=1}^{n} IPC(P_i, P_j)$$

5.6 Conclusion

In this chapter we propose a suite of coupling metrics to compute different types of coupling of UML designs (Package, Class and Sequence diagrams) stored in XMI format. The proposed suite of coupling metrics measures all the dimensions of coupling like inheritance, interaction and component coupling considering all the factors that exist in design. The Package coupling metrics give a package-level of design coupling. Figure 15 gives the overview of UML coupling suite.

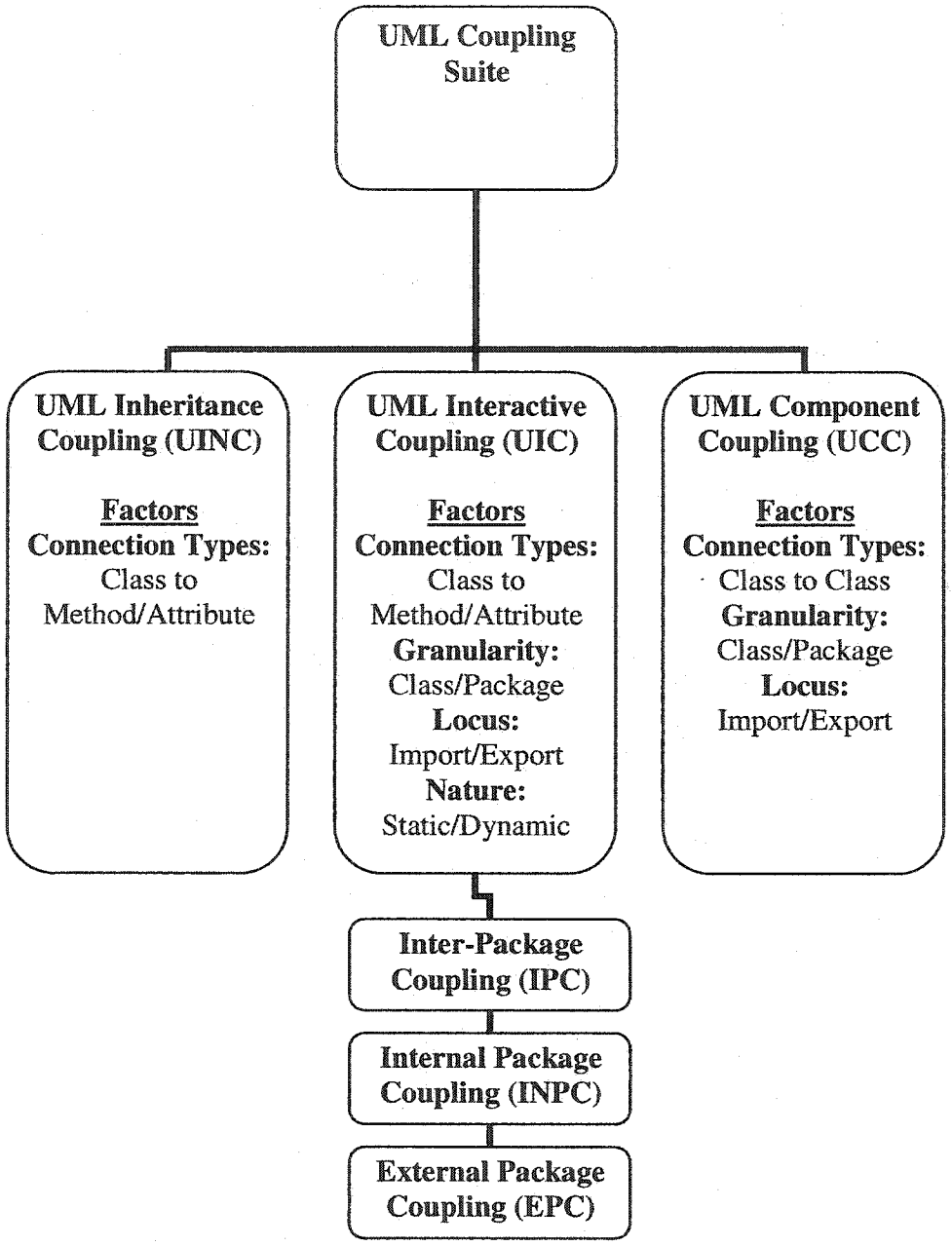

Figure 15: UML Coupling Suite

Chapter 6

UML Metric Tool

In this chapter, we discuss our prototype tool that we developed for measuring coupling. It also demonstrates an approach for computing coupling from UML designs. The approach can be easily extended to compute other higher level metrics that can prove to be useful in design phase. Different approaches for XMI processing are presented in the first section of this chapter, then our approach and its justification is presented. The architecture and design of tool follows next. We end this section with different ways in which tool can be used.

6.1 XMI Processing

In this section we present some of approaches that may be used for processing XML in general and XMI in specific, to get relevant data out of the XMI documents. The strength and weakness of each approach is presented along with the types of application they are suited for.

6.1.1 XML Query languages

One approach of XMI processing is to use a standardized XML query or transformation language. There are several such languages including the W3C proposed standard Extensible Stylesheet Language Transformations (XSLT). A more comprehensive literature about several approaches is W3C page on Query Languages at The Technology & Society Domain. These query languages provide a declarative SQL like interface to XML data. But given the differences in the many versions of the XMI (and UML) standards and the constant upgrade of these standards, this approach sounds like building from scratch with every change in the standards of UML and XMI.

6.1.2 XMI-Specific API's

Another approach is the use of APIs for XMI data management. Here the onus of complying with latest versions of standards will lie with the API implementations and we just need to shift to the latest API implementation. We discuss next three such APIs.

The Java Metadata Interface (JMI) Specification is based on the Meta Object Facility (MOF) specification from the Object Management Group (OMG), an industry-endorsed standard for metadata management. It enables the creation, storage, access, discovery, and exchange of metadata. JMI defines standard Java interfaces to these modeling components, and thus

enables platform-independent discovery and access of metadata. These interfaces are generated automatically from the models, for any kind of metadata. Additionally, metamodel and metadata interchange via XML is enabled by JMI's use of the XML Metadata Interchange (XMI) specification. NSUML and CIM Standard are reference implementations of JMI.

NSUML (Novosoft UML) metadata framework is based on JMI specification and generated classes that are required by JMI specification and also provides additional services like event notification, undo/redo support, and XMI support.

CIM Standard is Java implementation of the MOF and JMI. It is a platform independent metadata infrastructure for developing model-driven tool and application suites.

6.1.3 Criteria for Selection

There are following criteria for selection of an approach for XMI processing for the type of processing that we plan to do.

6.1.3.1 Simplicity

Any metrics extraction process as well as a metric cannot afford to be so complex to render it impractical for calculation. A major requirement is the ease and simplicity of implementation. In this XSLT scores low than other

approaches because XSLT is declarative language. Hence its learning curve is steep. Apart XMI in its present (1.0) form does not render itself easily for XSLT processing. Future versions of XMI are changing particularly changes like storing values in element attributes rather than element content can simplify XSLT processing. Reason for staying with lower versions of XMI is the discussed in next section. As for API's SAX, JAXP, DOM are well documented ones for XML, but NSUML the only one for specially XMI is not so well documented. Hence it is not straightforward to use, but the most important reason against its use is inter-operability that is discussed next.

6.1.3.2 Inter-Operability

As discussed in the introduction about the chaos with XMI standard, inter-operability is the one of the major criteria when working with XMI or UML. A metric tool can only collect data from UML models stored in XMI format, but these models will be generated in some case tools. Each case tool has its own peculiarities and whims. Lot has been written on the issues with case tools in these references, and if we add XMI support to our requirement most of case tools do not even stand in the race. In summary any approach should either be inter-operable with at least some tools and versions of XMI and UML or it should be flexible enough for upgrade that is our next criteria.

6.1.3.3 Flexibility

Using API's gives the flexibility if the API is 'alive' in the sense that it is supported by its makers and a strong user community. Only DOM and SAX satisfy this criterion.

6.1.3.4 Speed and Storage

Since size of XMI files is considerably large chosen approach for metric processing should be practical in terms of speed and memory that it can be used to process real project models. A benchmark study [77] compares the performance of all of the above XML processing approaches. It rates SAX as best in terms of time and memory.

Following table shows our subjective scores for each of the approach and justifies our choice in view of our objectives and scope.

Approach	Simplicity	Inter-Operability	Flexibility	Speed-Memory	Total Score
XSLT	1	1	2	0.5	4.5
DOM	2	2	2	1	7
NSUML	1.5	0.5	1	1	4
SAX	2	2	2	2	8

Hence we chose to implement a state machine-using SAX to capture all the information we need from XMI files and store it in an independent data model for further metric processing. The brief design and requirements of tool follows next.

6.2 Use case Specifications

XMI Parser is a part of another parent project "OOMeter". The main objective of this project is a metrics tool "OOMeter" that could capture all metrics for a system for analysis and also support customization of metrics. Below we show the use cases of OOMeter in general and XMI Parser in specific and how they fit in the overall picture. Further details of OOMeter can be found in its Project Report.

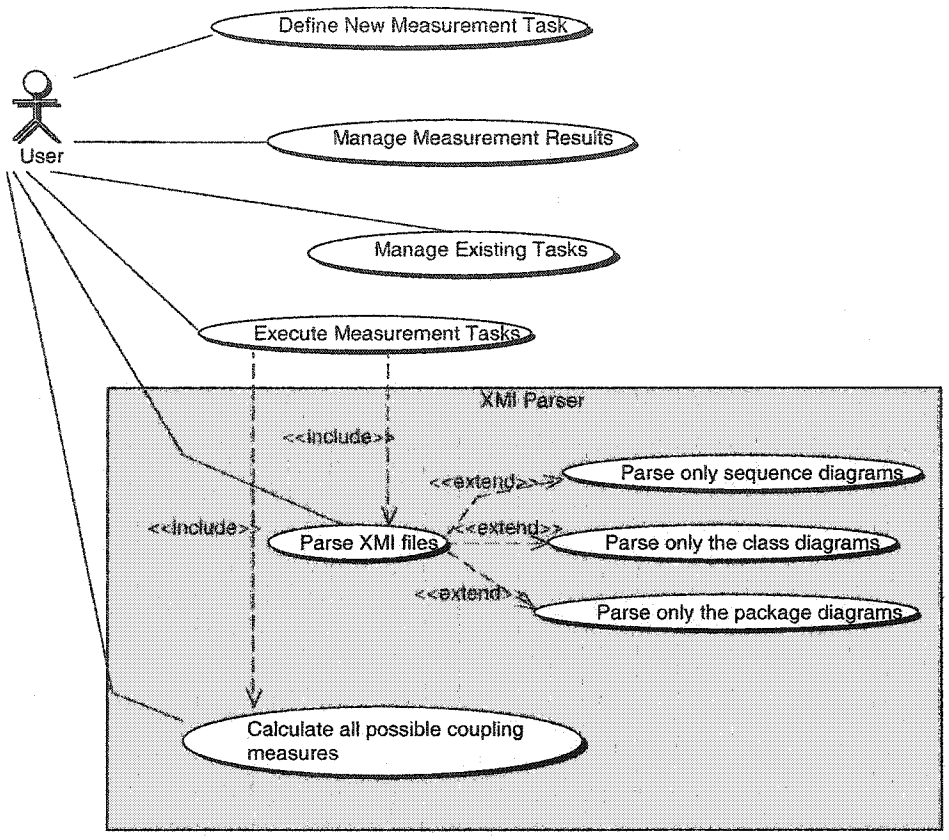

Figure 16: Requirement XMI Parser

The main functional requirements of XMI Parser were

- The state machine should be able to extract all information related to classes, attributes, methods and there interrelations, including package structure.
- It should compute important proposed coupling metrics transparently and accurately

The important non-functional requirements of the design are:

- Flexible so that changes of XMI can be accommodated
- It should be robust enough to ignore the whims of case tools (as discussed before)

6.3 Architecture

Architecture of OOMeter can be classified as a heterogeneous architecture in Garlan & Shaw's [56] terminology. Main components are two parsers for java and XMI, two data repositories for storing source data and metrics output and a front end.

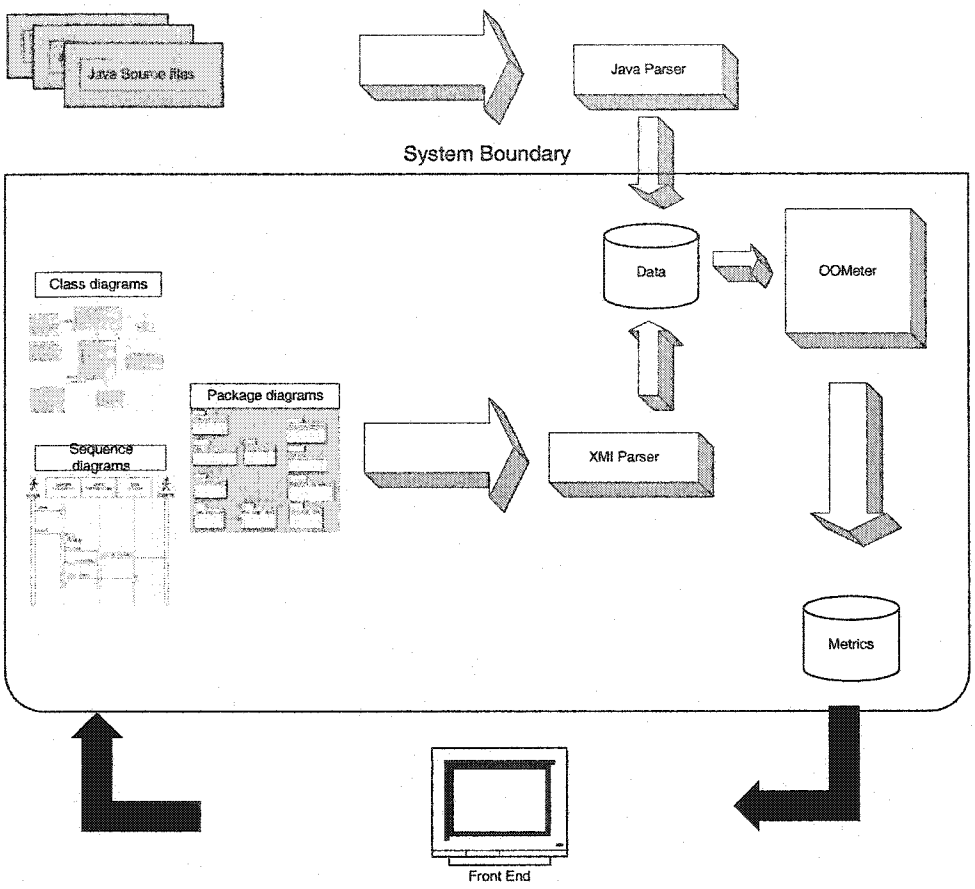

Figure 17: Architecture OOMeter

The system boundary in the above architecture shows our main focus of discussion. The results of XMI parsing are stored in the data repository in language independent format. It can be observed in the data model shown in Figure 18 that it captures all basic information that is needed for most of the software metrics.

The data model serves as our language independent repository for storing all the information parsed from the projects, whether it may be java source files or UML models stored in XMI.

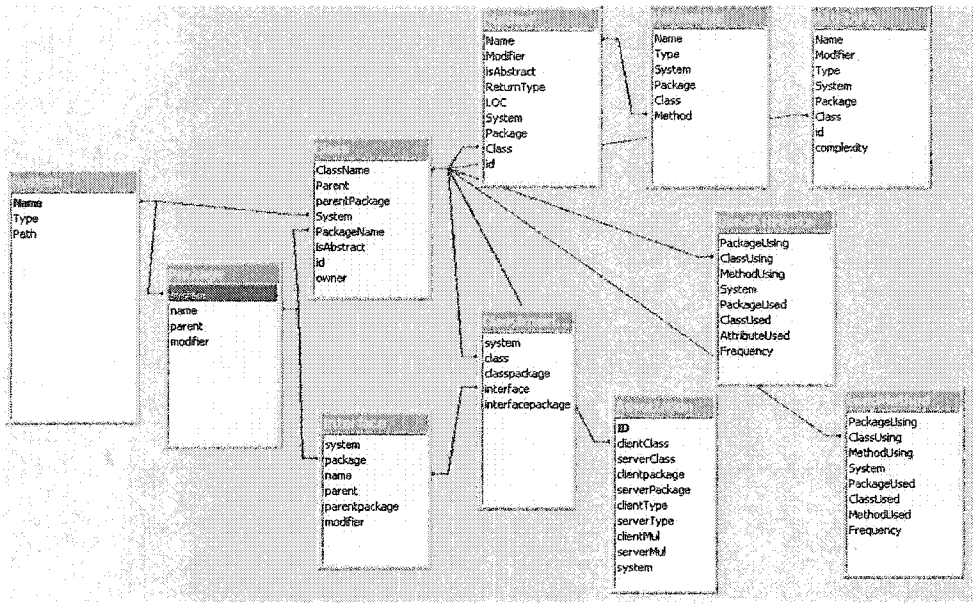

Figure 18: Data Model of OOMeter

Figure 19 shows the component or package diagram of OOMeter. *OOMeter* is the main package of the system that uses other packages. *XMIParser* and *JavaParser* are the parsers for XMI and java files respectively. *Common* contains the database handling classes and *XMIParserTest* and *JavaParserTest* are the Junit test classes for the two parsers.

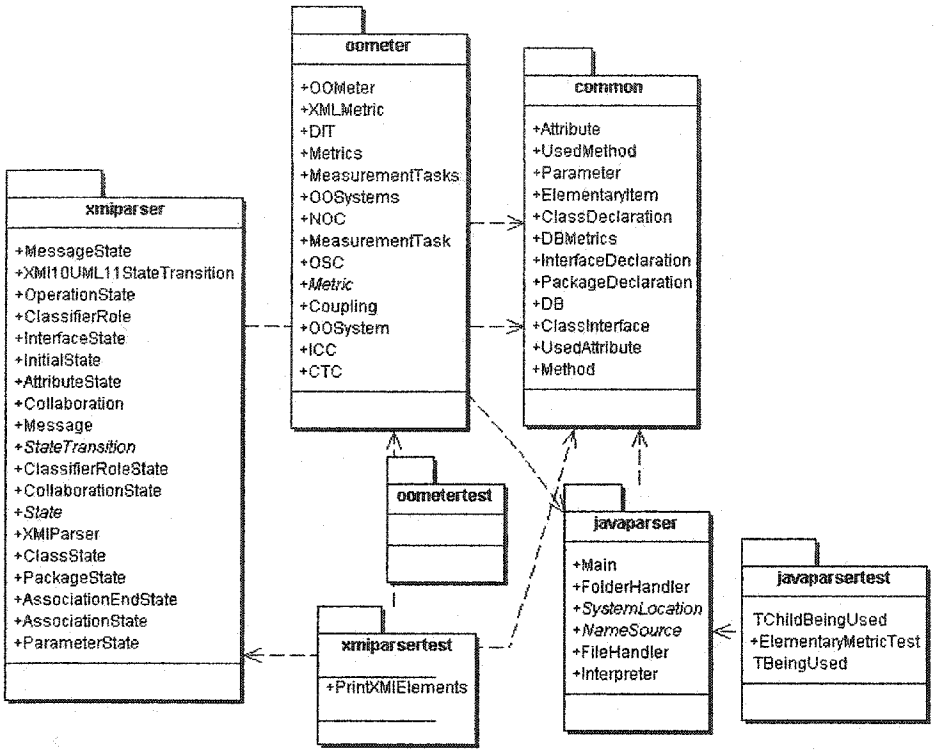

Figure 19: Component Diagram of OOMeter

6.4 Design of XMI Parser

In this section we will focus on the design of the XMI parser component of the system, Appendix provides the detailed design of tool. As discussed in the XMI processing approach selection section, we selected a state machine-using SAX. We used a hybrid of table driven state pattern and sub classing state pattern [57]

The XMI parser goes through several states that denote the basic entities in the UML Meta model, capturing in each state all the information needed from the child nodes and entity values.

The following state charts show the transitions for each of the UML diagram types that we take as input.

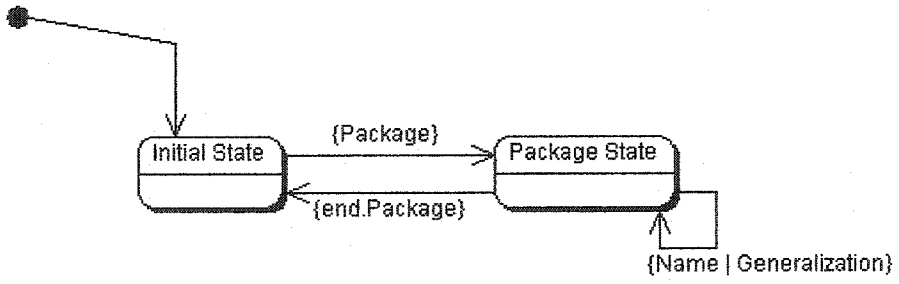

Figure 20: State Chart for Package diagram

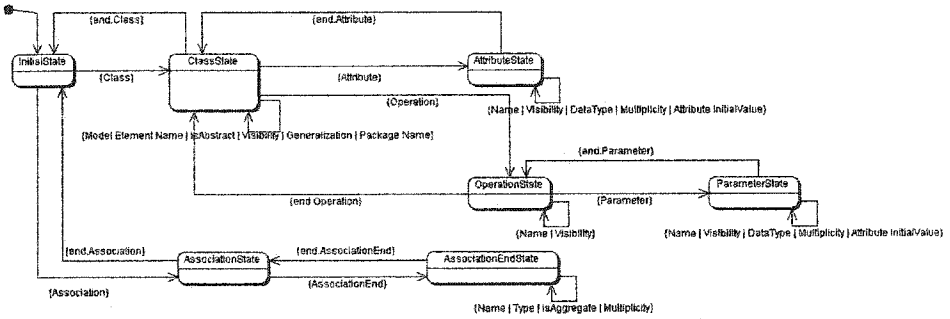

Figure 21: State Chart for Class diagram

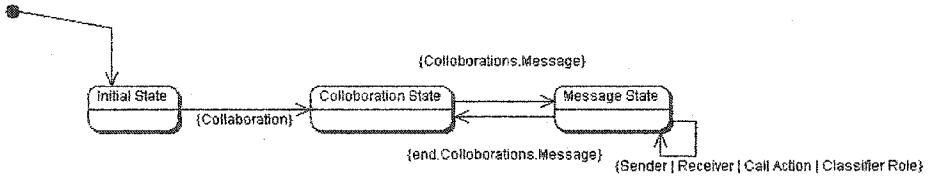

Figure 22: State Chart for Sequence diagram

The class diagram below shows all classes of XMI Parser.

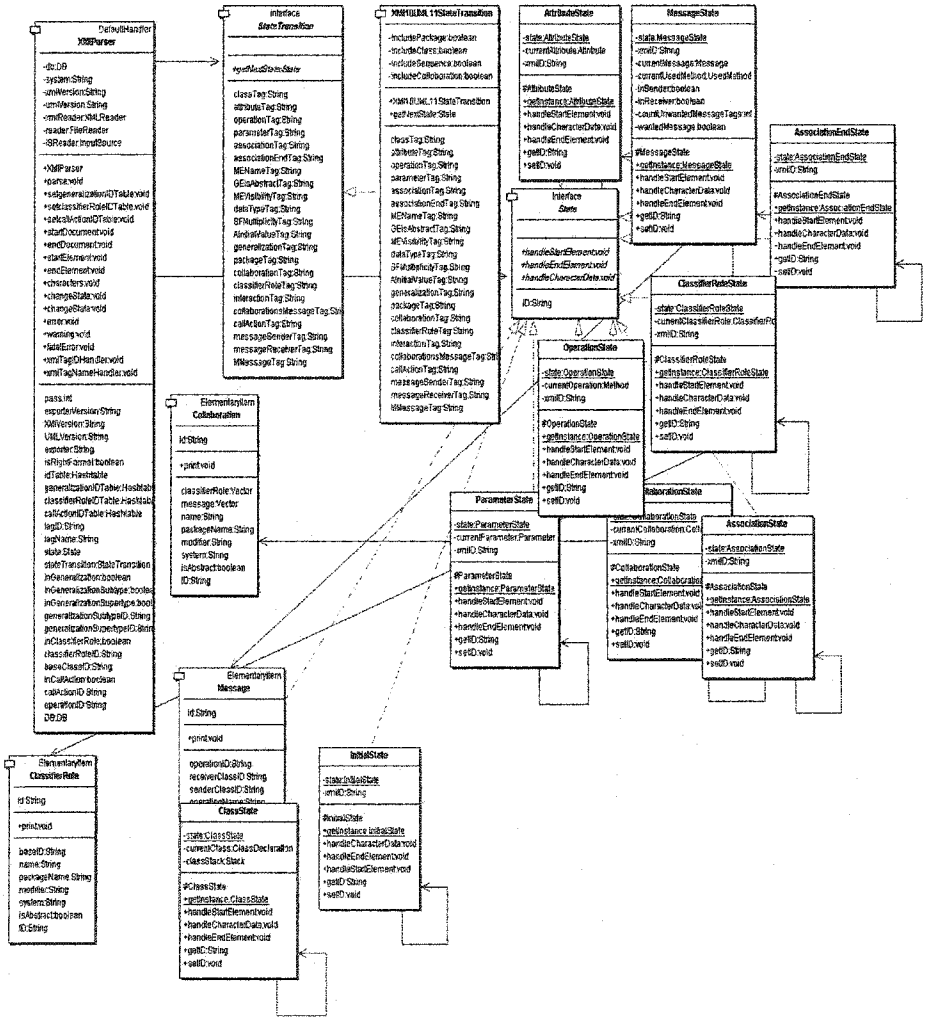

Figure 23: Class diagram XMI Parser

As show in the class diagram each of the concrete state implements the *state* interface. All the logic of state transitions is encapsulated in XMI and UML version specific State Transition table *XMI11UML10StateTransition* in this case. This transition table implements the *StataTransition* interface; hence if we want to support another version of XMI or UML other concrete classes can be derived for those versions. The amount of change needed in concrete classes depend on the nature of change in version. For example node names can be easily mapped from XMI 1.0 to XMI 1.1, hence change is a question of mapping the tag names, but for major changes as in XMI 2.0, this may not be very straightforward.

6.5 XMI Parser Metric Tool

Table 12 compares XMI Parser with other metric tools covered in section 2.4.4.

Table 12: Comparison of XMI Parser with other UML metric tools

Name of Tool	Metrics calculated	Approach used	Limitation	Reference
UML Model Comparer	Four Similarity Metric	Semantic distance measures and class hierarchy structure	Tool need optimization as the process is heavily compute intensive.	[93]
MAISA	Architectural Pattern measures: Number of	CSP Pattern mining using Java and Prolog for pattern	Work is significant from design pattern view, but not from metrics, because it includes only	[81]

Name of Tool	Metrics calculated	Approach used	Limitation	Reference
	classes Number of messages Depth of inheritance hierarchy Predictive Estimate over final systems based on pattern mining analysis: Size, performance, complexity	representation	elementary metrics.	
Metrics from UML	Global and Class level metrics. In global metrics it gives numbers (min, max, avg, percentage) of class, attribute, method, interfaces, associations in the system In Class level metrics it gives same numbers per class.		Work is significant from XMI processing view, but not from metrics, because include elementary metrics. Limitations from XSLT for XMI processing are also visible, as discussed in Chapter on Implementation.	[78]
OSMAT	Including the above it also computes: CK suite Weighted method per class Number of children per		Includes only elementary metrics.	[79]

Name of Tool	Metrics calculated	Approach used	Limitation	Reference
	class			
Cool UML	Similar to "Metrics from UML", but excluding associations		Includes only elementary metrics.	[80]
SDMetrics	Exhaustive elementary metrics and some composite metrics	XSLT	Lacks some composite metrics	[96]
XMI Parser	Elementary metrics and composite metrics like coupling metrics with different granularities	SAX	Lacks some composite metrics	

Chapter 7

Validation

In this chapter, we present an evaluation of design level coupling metrics. We theoretically validate the proposed metrics. Section 7.2 presents this theoretical validation. We empirically validate the UML design coupling metrics against the source level coupling metrics and other design level coupling metrics proposed in literature. Section 7.3 introduces the goal and hypothesis of this validation. Section 7.4 presents the experimental plan to achieve the goal of testing the stated hypothesis about design coupling metrics. In Section 7.5 we present the results of empirical validation. In section 7.6 we discuss the results.

7.1 Theoretical Validation

Representation Condition of measurement asserts that a metric must map entities into numbers and empirical relations into numerical relation in such a way that the empirical relations preserve and are pre-served by the numerical relations [101]. Coupling metrics have been theoretically validated using modified Weyuker's properties [114][37] and Briand's Framework [46]. Weyuker [114][37]proposed nine properties that should be satisfied by any

complexity metric. Out of these Chidamber et. al. [37] dropped three properties as they did not apply to object oriented metrics. These properties are:

1. **Noncoarseness**: Given a class P and metric u another class Q can always be found such that $u(P) \neq u(Q)$.
2. **NonUniqueness**: There can exist distinct classes P and Q such that $u(P) = u(Q)$
3. **Design Details are Important**: Given two class designs, P& Q, which provide the same functionality, does not imply that $u(P) = u(Q)$. The specifics of class must influence the metric value.
4. **Monotonicity**: For all classes P and Q, $u(P) \leq u(P+Q)$ and $u(Q) \leq u(P+Q)$, where P+Q implies combination of P& Q.
5. **Nonequivalence of Interaction**: $u(P) = u(Q)$ does not imply that $u(P+R) = u(Q+R)$.
6. **Interaction increases Complexity**: When two classes are combined the interaction between them can increase the metric value.

Similarly Briand [46] also proposed five intuitive coupling properties that any coupling measure should satisfy. OuterR(c) is defined as total of export and import coupling and InterR(C) = $\bigcup_{c \ni C} OuterR(c)$.

1. **Nonnegativity**: The coupling of a class can never be negative.

$$Coupling(c) \geq 0 | Coupling(C) \geq 0$$

2. **Null Value**: The Coupling of a class is null if OuterR(c) or InterR(C) is empty.

$$OuterR(c) = \phi \Rightarrow Coupling(c) = 0 | InterR(C) = \phi \Rightarrow Coupling(C) = 0$$

3. **Monotonicity**: If we add something to a class, its coupling should only increase.

$$Coupling(c) \leq Coupling(c) = 0 | Coupling(C) \leq Coupling(C')$$

4. **Merging of connected classes**: Merging of two related classes can only decrease coupling.

$$Coupling(c_1) + Coupling(c_2) \geq Coupling(c') | Coupling(C) \geq Coupling(C')$$

5. **Merging of unconnected classes**: Merging of two unconnected classes should not affect coupling at all.

$$Coupling(c_1) + Coupling(c_2) = Coupling(c') | Coupling(C) = Coupling(C')$$

Briand's properties are closer to intuition of coupling than Weyuker's because they apply in general to all complexity metric, where as Briand's properties apply specifically to coupling metric. Briand also showed that come of the Weyuker's properties go against the intuition of coupling, like property 6. Hence we will theoretically validate our proposed metrics against Briand's properties.

For Size metrics, Briand proposed three properties:

Briand [108] published following three intuitive properties that any size measure should satisfy.

Nonnegativity: The size of a module can never be negative.

$$Size(c) \geq 0 | Size(C) \geq 0$$

Null Value: The size of a module is null if it is empty.

$$S =< E, R >$$
$$E = \phi \Rightarrow Size\ (S) = 0$$

Disjoint Module Additivity: The total size of system consisting of two disjoint modules should be sum of size of individual sub components.

$$m1 \subseteq S, m2 \subseteq S, E = E_{m1} \cup E_{m2}, E_{m1} \cap E_{m2} = \phi$$
$$Size(S) = Size(m1) + Size(m2)$$

7.1.1 Theoretical Validation of UML Class Size

Nonnegativity: In the equation for class size each of the term except number of messages depends on the attribute size that is always positive (Table 7). Finally number of messages cannot be negative.

$$Size(c) = \sum_{i=1}^{n} Size(att_i) + \sum_{i=1}^{n} Size(meth_i) + \sum_{i=1}^{n} Size(innerClass_i)$$

Hence the right hand side of above equation of class size can never be negative this property is proved.

Null value: If a class is empty it implies that it has no attributes and no methods (and messages), hence the class size is null.

Disjoint Module Additivity: If two classes are merged then methods and attributes complexities will be added, and hence over all size will be sum of individual sizes.

7.1.2 Theoretical Validation of UML Interactive Coupling Metric

Nonnegativity: In the equation for computing the entries of definition matrix the right hand side represents class size that can never be negative.

$$d_{ij} = \sum MethodSize(uses(i, j))$$

Null value: Suppose we have a system in which import and export interaction coupling (InterR(c)) is null, it implies that there is no interaction

connections among the classes of system. Hence the initial definition matrix will contain a zero value for all cells resulting in a zero overall coupling.

Monotonicity: A method, attribute or an inner class can be added to a class. We discuss effect of each:

- If an attribute is added to a class, it will increase the class size. This will increase its coupling with other classes (where it is server or expertly coupled)
- If a method is added to a class, it may add a connection thereby increasing coupling or will not add any connections. But still in this case also it will increase the size of class. This will increase its coupling with other classes (where it is server or expertly coupled)
- If an inner class is added to a class, it will increase the class size. This will increase its coupling with other classes (where it is server or expertly coupled)

In all above cases the coupling may only increase but never decrease.

Merging of connected modules: If two connected classes are merged then some of the connections that were present between these classes will vanish decreasing their coupling. Hence over all coupling will be maximum of individual connections.

Merging of unconnected modules: If two unconnected classes are merged then the connections in the system will remain unchanged. No new connections will appear, hence over all coupling will remain unchanged.

7.1.3 Theoretical Validation of UML Component Coupling Metric

Nonnegativity: In the equation for component coupling, it can be seen that it is the summation of class size that can never be negative.

$$d_{ij} = \sum ClassSize(association(i, j))$$

Null value: In a system where the import and export component coupling (InterR(c)) is null, it implies that there are no associations among the classes of the system. Hence the initial definition matrix will contain a zero value for all cells resulting in a zero overall coupling.

Monotonicity: A method, attribute or an inner class can be added to a class. We discuss effect of each:

- If an attribute is added to a class, it will increase the class size. This will increase its coupling with other classes (where it is server or expertly coupled)

- If a method is added to a class, it will have no effect on associations but it will increase the size of class. This will increase its coupling with other classes (where it is server or expertly coupled)
- If an inner class is added to a class, it will increase the class size. This will increase its coupling with other classes (where it is server or expertly coupled)

In all above cases the coupling may only increase but never decrease.

Merging of connected modules: If two connected classes are merged then some associations may vanish thereby reducing the coupling. Hence over all coupling will be maximum of the old and modified system.

Merging of unconnected modules: If two unconnected classes are merged then association connections will remain unchanged. No new connections will appear, hence over all coupling will remain unchanged.

7.1.4 Theoretical Validation of UML Package Coupling Metric

Nonnegativity: Since the package coupling is just sum of inter-class coupling that has already been shown to satisfy this property, hence this property is satisfied.

Null value: In a system where the import and export package coupling (InterR(c)) is null, it implies that there are no coupling connections between

classes of different packages. Thus this implies the inter class coupling matrix was itself null and resulting in a zero overall package coupling.

Monotonicity: If a class is added to a package it may either introduce new connections thereby increasing coupling or it will have no effect on coupling. In both the cases the coupling may only increase but never decrease and same will be the case with overall coupling.

Merging of connected modules: If two connected packages (i.e. with some non zero coupling value between them) are merged then that coupling will disappear thereby reducing the overall coupling of system. Hence this property can not be violated.

Merging of unconnected modules: If two unconnected packages (i.e. with zero coupling between them) are merged then the coupling of new system will remain unchanged, hence this property can not be violated.

7.2 Goals and Hypothesis for Empirical Validation

The goal of this empirical validation is to validate whether the design level metrics measure convey similar information as their source level counterparts, and compare them with similar design level measures, if any.

The hypothesis for this validation can be stated as follows:

- Hypothesis H1: The design level metrics correlate with their source level counterparts.
- Hypothesis H2: The design level metrics correlate with other design level metrics, if any.

7.3 Experimental Design

The metrics to be validated are elementary metrics that are used for computing coupling and inheritance based coupling metric, interaction coupling metric and Package coupling metric. For some design metrics we were unable to find their exact counterparts, hence we selected the closest metrics that measures similar properties. Priority was given to those metrics that have already been internally as well as externally validated. Table 13 shows the experimental plan for validation. For Inheritance coupling metrics, the results of previous case study that computed the same metric from source code was available, hence those case studies were repeated for these metrics. Please note that design level counterpart to class size are computed by SDMetrics, like Number of attributes, methods etc. But since we had already source level metrics to compare with, we neglected these elementary metrics that give a very rough idea of class complexity.

Table 13: Experimental Plan

Metric to be Validated	**Case Study**	**Source Level Counterpart**	**Design Level Counterpart**	**Number of Experiments**	**Hypothesis to be tested**
Class Size	OOMeter	LOC WMAC1 WMAC2 Cyclomatic Complexity		4	H1
Inheritance Coupling Metric	Java Lang.Ref & Awt.Event	AlMulla IC		2	H2

Metric to be Validated	**Case Study**	**Source Level Counterpart**	**Design Level Counterpart**	**Number of Experiments**	**Hypothesis to be tested**
	OOMeter		HenriLi	1	H1
Import Interaction Coupling Metric	OOMeter	VOD	MsgSent	2	H1 and H2
Internal Package Coupling Metric	OOMeter		MessageSent Within	1	H2
External Package Coupling Metric	OOMeter		MessageSent Outside	1	H2

In this section, we present an empirical validation of the design metrics. Since the higher-level composite metrics like class coupling and package coupling are based on elementary metrics like attribute, method and class complexity, hence we first validate them using our own tool "OOMeter" as a case study. The reason for selecting this was that since we developed the system, we were aware of the details and its design issues.

7.3.1 Treatments (Design Coupling Metrics)

The treatments that are applied to experimental objects (UML Package, Class, Sequence and Collaboration diagrams, Java Source Code) are the coupling metrics implemented in XMI Parser and the tools Together and SDMetrics.

7.3.2 Experimental Objects

The experimental objects are

- OOMeter UML Design (.xmi files for package, class, sequence and collaboration diagrams) and Source code (.java files). OOMeter is software metrics tool under development that can compute the metrics from java source programs or UML designs. It consists of 53 classes and 6 packages. The code of consists of about 6000 LOC (Lines of Code).

- Java.Lang.Ref Package of Java 1.2 UML Design (.xmi files for package, class, sequence and collaboration diagrams) and Source code (.java files). Java.Lang.Ref Package is a package in JDK that consists of 7 classes.
- Java.Awt.Event Package of Java 1.2 UML Design (.xmi files for package, class, sequence and collaboration diagrams) and Source code (.java files). Java.Awt.Event Package is a package in JDK that consists of 23 classes.

The UML designs for each of the above experimental objects were reverse engineered from the java source code using Together case tool and exported in XMI format. Class and Package diagrams are reverse engineered by default in Together, but Sequence diagram have to be explicitly reverse engineered.

7.3.3 Subjects

Since the whole process from reverse engineering to treatment application is completely automated by the case tools and metrics tools the effect of subject bias in experiments is not a consideration.

7.3.4 Data Collection and Validation Procedures

Metrics Results from XMI Parser are automatically saved in the Excel Workbook using JExcel API. For each metric computed at least three sheets are created to save the metrics at the finest granularity as well as the summary statistics and relative and normalized values. The intermediate data structures used for coupling computation like class hierarchy are also stored in the workbook just to verify the results. This verification was done manually initially using small test programs. The results obtained from tools like Together and SDMetrics were also saved manually in the same excel workbook.

7.3.5 Data Analysis Procedure

For both the hypothesis we obtain the two metrics to be compared and test the two metrics for correlation using Spearman Rank Correlation test [107][113]. The Spearman Rank Correlation was used because the coupling measure is ordinal. The Spearman Rank Coefficient is computed as follows:

$$SpearmanRankCorrelationCoefficient = 1 - \frac{6\sum d^2}{n(n^2-1)}$$

where d is the difference between ranks of two respective observations from the two datasets and n is the number of ranks.

Note that Null hypothesis below means the converse of the hypothesis that we want to prove.

For the class size we use the Pearson Correlation because the size measure is on ratio scale. The Pearson Coefficient is computed as follows:

$$PearsonCorrelationCoefficient = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{(\sum X^2 - \frac{\sum X^2}{N})(\sum Y^2 - \frac{\sum Y^2}{N})}}$$

where X and Y are the data sets and N is the number of cases.

We calculate the two-tailed p-value that is the probability under the null hypothesis to obtain a result as extreme as the observed one, at the two tails of the distribution (meaning that the result obtained was from randomness of sample and not real correlation). We reject the null hypothesis when the probability is lower than the alpha level that is the probability that our results are false.

The value of $\alpha = 0.05$ i.e. Confidence Interval of 95 %. (We can say with 95 % confidence that results are true). We used the XLSTAT [115] package that simplifies the procedure of computing the p-value [113].

7.4 Validation results

To validate the UML metrics we used the already validated metrics from literature and implemented by most of the metric tools.

7.4.1 UML Class Size

For validating UML class size we used the metrics, LOC (Lines of Code), WMC (Weighted Method per Class), and Cyclomatic Complexity. One of the tools that measured these metrics reliably was found to be Together [93]. The results from Together are in the appendix. (Table 28)

Note that Together calculates two variations of WMC metric; WMC1 in which the methods are weighed by their Cyclomatic complexity and WMC2 where the method are weighed by their number of parameters and return value.

The results for our class size metric are also shown in appendix. (Table 29)

The results of correlation of our metric with each of the selected metrics for class complexity are shown in Table 14 along with the correlation values.

Figure 24 shows the scatter diagram of correlation of UML class size metric with LOC metric

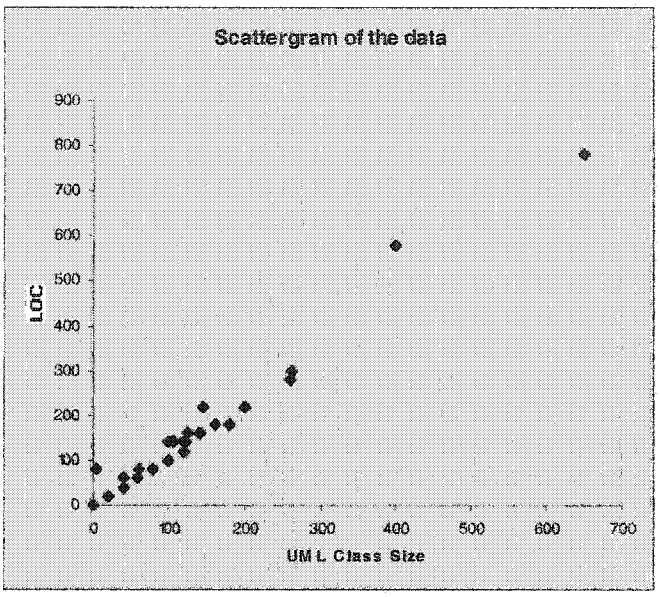

Figure 24: Pearson Correlation Scatter Chart for LOC Vs UML Class Size

Table 14: Pearson Correlation results for LOC Vs UML Class Size

Pearson Correlation Coefficient	0.973
N	53
p-value	0.0001
Alpha	0.05

The correlation value is close to 1 that shows a strong correlation between UML Class size and LOC.

For the significance of this correlation (to test whether this value is from randomness of sample) we compare the p-value with the alpha value. Since

p-value is less than the alpha value we conclude that the correlation is significant at the confidence level of 95 %.

Figure 25 shows the scatter diagram of correlation of UML class size metric with Cyclomatic complexity metric.

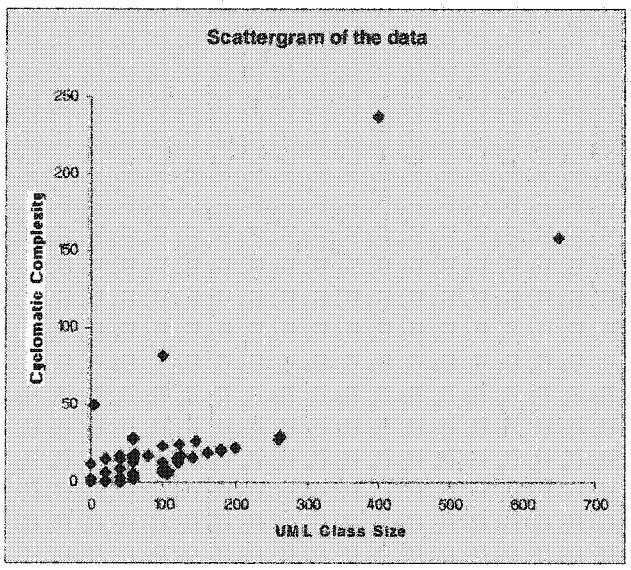

Figure 25: Pearson Correlation Coefficient Scatter Chart for Cyclomatic Vs UML Class Size

The strength of correlation in this case is lower because of two reasons:

Cyclomatic complexity is not exactly a size measure.

The Cyclomatic Complexity metric is defined for methods, when together computes it for class, it sums up all cyclomatic complexities in the class to give the class level metric.

Table 15: Pearson Correlation Coefficient results for Cyclomatic Vs UML Class Size

Pearson Correlation Coefficient	0.748
N	53
p-value	0.014
Alpha	0.05

Now we compare the p-value with the alpha probability. Since p-value is less than the alpha value, we conclude that the observed correlation is significant at the confidence level of 95 %.

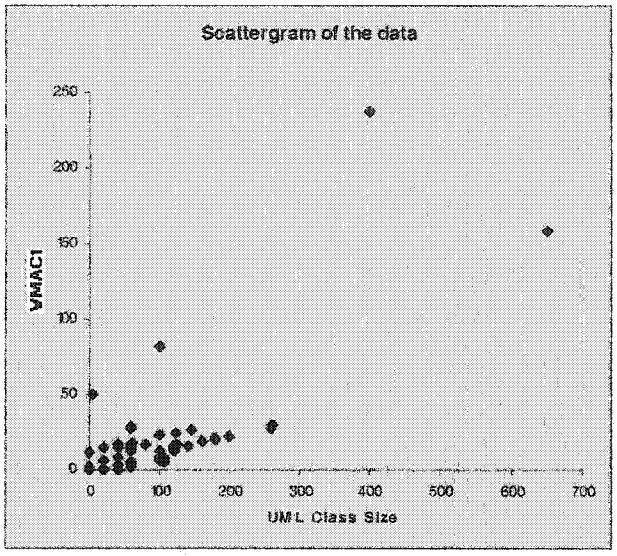

Figure 26: Pearson Correlation Coefficient Scatter Chart for WMC1 Vs UML Class Size

Figure 26 shows the scatter diagram of correlation of UML class complexity metric with WMC1 complexity metric

Table 16: Pearson Correlation Coefficient results for WMC1 Vs UML Class Size

Pearson Correlation Coefficient	0.748
N	53
p-value	0.0001
Alpha	0.05

Here also strength of correlation is relatively lower. This is because WMC is a source level size metric that depends on weights of method body. But the method body is not available in design stage (in UML).

Since the p-value is less than the alpha value, we conclude that this observed correlation is significant at the confidence level of 95 %.

Figure 27 shows the scatter diagram of correlation of UML class complexity metric with WMC2 complexity metric

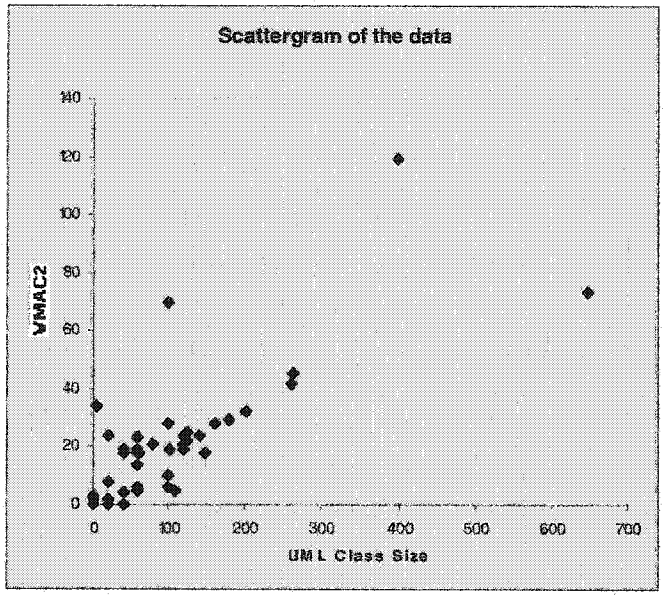

Figure 27: Pearson Correlation Scatter Chart for WMC2 Vs UML Class Complexity

Table 17: Pearson Correlation results for WMC2 Vs UML Class Complexity

Pearson Correlation Coefficient	0.737
N	53
p-value	0.0001
Alpha	0.05

Here also strength of correlation is lower because of reason discussed before.

Since the p-value is less than the alpha value we conclude that the observed correlation is significant at the confidence level of 95 %.

Results for UML Class Size show that our design level measure for class size correlates with most of the existing standard metrics of class size at Source Level. Hence it can be used as a reliable metric for computing other composite and higher level metric. It can also be used to estimate the size, effort and cost of project from the initial design.

7.4.2 Inheritance Coupling

Using the same case study we compare the UML Inheritance coupling metric with the one obtained from source code.

We use these case studies because the Source Inheritance Metrics were computed from them, and we wanted to compare the design metrics with

source metrics. For comparison with design level metric we use third case study "OOMeter".

7.4.2.1 Case Study 1: Java.Lang.Ref Package

Figure 28 shows the simplified class diagram of this case study that is a package from Java 1.2. It consists of seven classes and two inner classes.

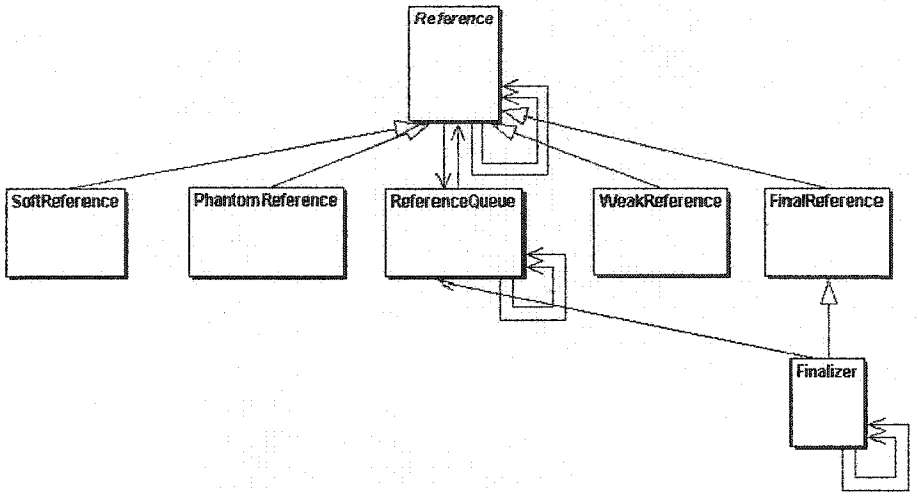

Figure 28:Case Study 1: Java.Lang.Ref Package

The results of inheritance metrics for this case study obtained by Elish [27] are shown in Table 18. Table 19 shows the results of the UIC (UML Inheritance Coupling) metric for this case study.

Table 18: Case Study 1 Elish Results

Class	Inheritance Coupling
java 1.2 Lang-Reference	0.6769
java 1.2 Lang-PhantomReference	0.7956
java 1.2 Lang-SoftReference	0.7644
java 1.2 Lang-WeakReference	0.7956

Table 19: Case Study 1 Design Metrics results

Class	Inheritance Coupling
java 1.2 Lang-Reference	0.306766917
java 1.2 Lang-PhantomReference	0.84962406
java 1.2 Lang-SoftReference	0.606874329
java 1.2 Lang-WeakReference	0.84962406
java 1.2 Lang- ReferenceQueue	0
java 1.2 Lang- Finalizer	0.349641499
java 1.2 Lang- FinalReferencee	0.758279602

The results of Spearman Rank Correlation are show in Figure 29.

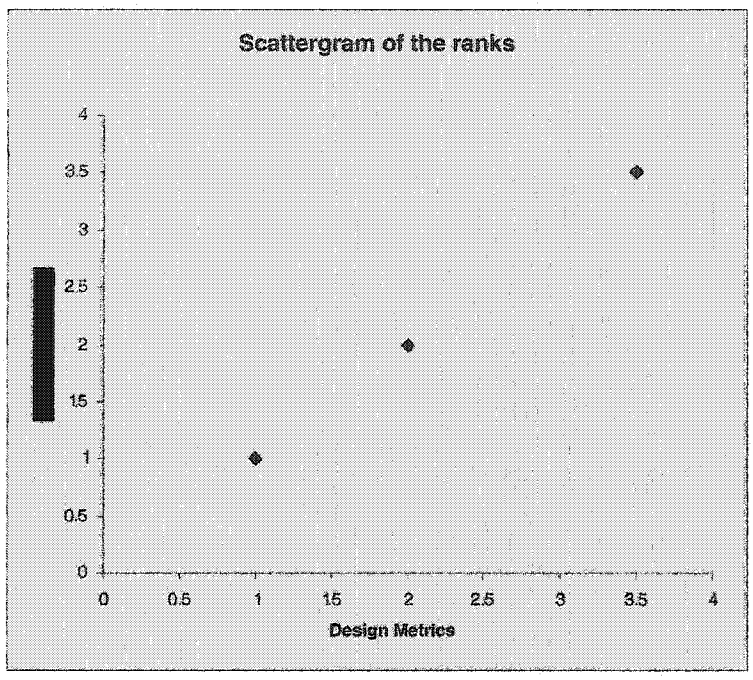

Figure 29: Case Study 1 Spearman Rank Correlation Scatter Chart for Case Study 1

Table 20: Case Study 1 Spearman Rank Correlation results for Case Study 1

Spearman Rank Correlation Coefficient	0.957
N	7
p-value	0.0001
Alpha	0.05

Here the strength of correlation is nearly one showing that inheritance metric correlates well with source level metric.

Since p-value is less than the alpha-value, we conclude that the correlation is significant at the confidence level of 95 %.

7.4.2.2 Case Study 2: Java.Awt.Event

The second case study was Event Package of Java 1.2 that contains 23 classes as shown in Figure 30.

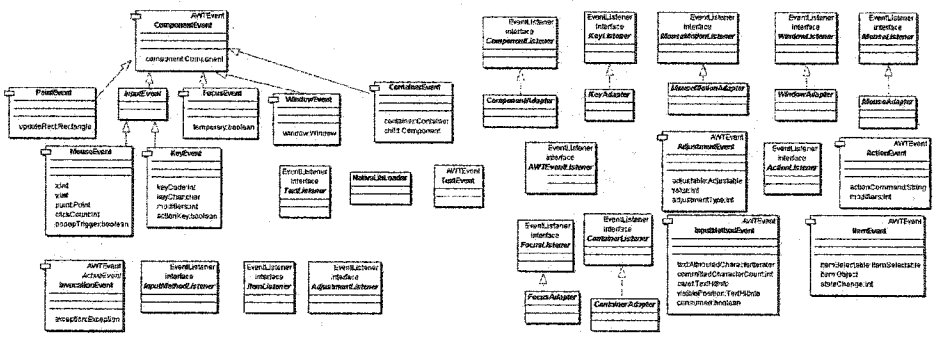

Figure 30: Case Study 2 Class Diagram

The results of inheritance metrics for this case study obtained by Elish [27] are in the appendix.

The reason for difference in the number of classes is that in previous work some of the classes were not captured and also the inner classes were ignored. Hence in further calculations for correlation we also ignored those classes. The results of Spearman Rank Correlation are show in Table 21.

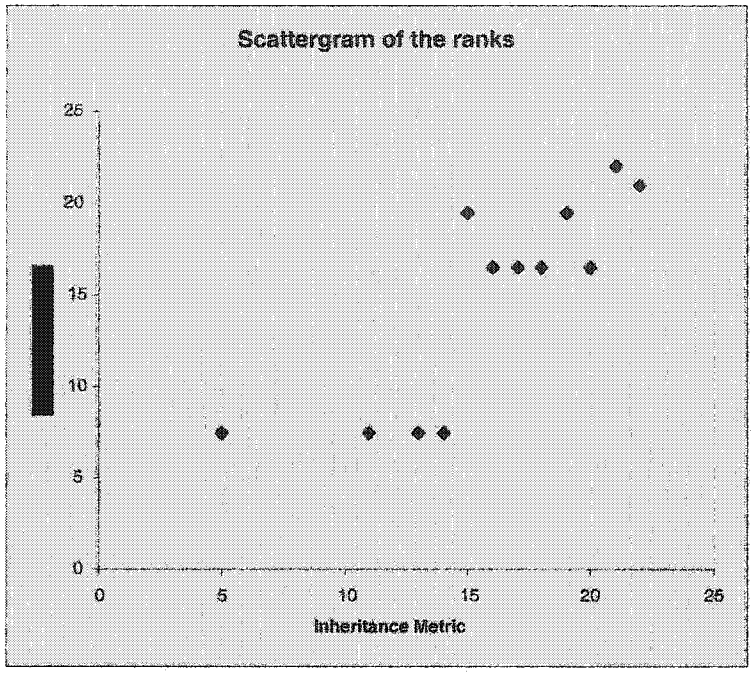

Figure 31: Spearman Rank Correlation Scatter Chart for Case Study 2

Table 21: Spearman Rank Correlation results for Case Study 2

Spearman Rank Correlation Coefficient	0.868
N	23
p-value	0.0001
Alpha	0.05

Strength of correlation in this case indicate strong correlation between two inheritance metrics.

Since p-value is less than the alpha-value, we conclude that the correlation is significant at the confidence level of 95 %.

7.4.2.3 Case Study 3: OOMeter

The last case study was OOMeter. The results of inheritance metrics for this case study are in the appendix.

The results of Spearman Rank Correlation are show in Figure 32.

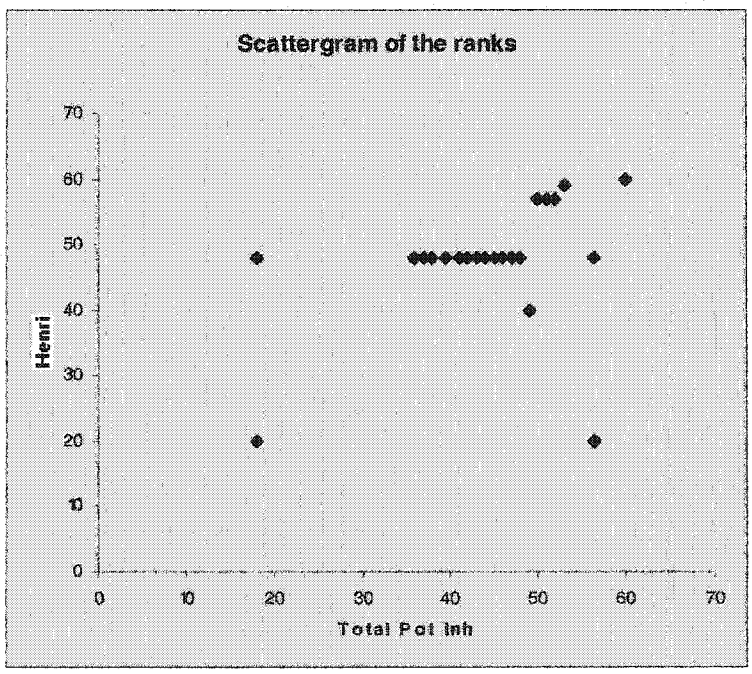

Figure 32: Spearman Rank Correlation Scatter Chart for Case Study 3

Table 22: Spearman Rank Correlation results for Case Study 3

Spearman Rank Correlation Coefficient	0.683
N	53
p-value	0.0001
Alpha	0.05

Strength of correlation shows medium correlation between HenriLi and UML Inheritance metric.

Since p-value is less than the alpha-value, we conclude that the correlation is significant at the confidence level of 95 %.

7.4.3 Interaction Coupling

The Interaction coupling that we used in the experiment is of type "Total Interaction Coupling" i.e. Sum of Import (where the class is coupled as a client to another class) as well as Export Coupling (where the class is coupled as a server to another class). We compared this metric against two sets of metrics; design level metrics and source level metrics. Together measures some source level metrics like VOD (Violation of Demeter Law)

and CBO (Coupling between Objects). SDMetrics [96] also measures some coupling measures from design like Message Based coupling, dependency client, etc. Exact definition of these metrics is not available, except for the standard metrics. Others are just counts of elementary components of UML design. These metrics also do not differentiate in locus of coupling (export or import). MsgSend Metric of SDMetrics is defined as number of messages sent to a class. Other metrics computed by these tools are mostly zero, which may indicate that they have not been implemented yet. We present the results obtained from these tools for the case study.

The results of Spearman Rank Correlation of UML Interaction coupling metric against source level metric VOD and design level metric MsgSent are show in Figure 33.

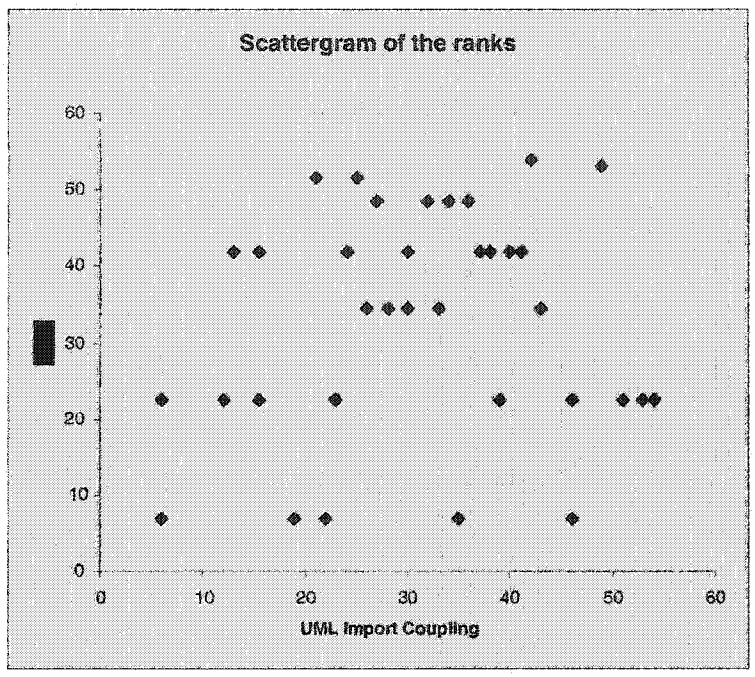

Figure 33: Spearman Rank Correlation Scatter Chart for UML Interactive Coupling Vs VOD

Table 23: Spearman Rank Correlation results for UML Interactive Coupling Vs VOD

Spearman Rank Correlation Coefficient	0.385
N	53
p-value	0.015
Alpha	0.05

Here we observe that correlation is weak. As we already noted in 4.4 that Interactive coupling from UML can only be approximately computed as compared with that from source. In sequence diagrams we do not consider all messages, but only the cross-class messages. Hence this correlation value was expected to be weak.

Since p-value is less than the alpha-value, we conclude that the correlation is significant at the confidence level of 95 %.

The results of Spearman Rank Correlation of UML Interaction coupling metric against design level metric MsgSent are shown in Figure 34.

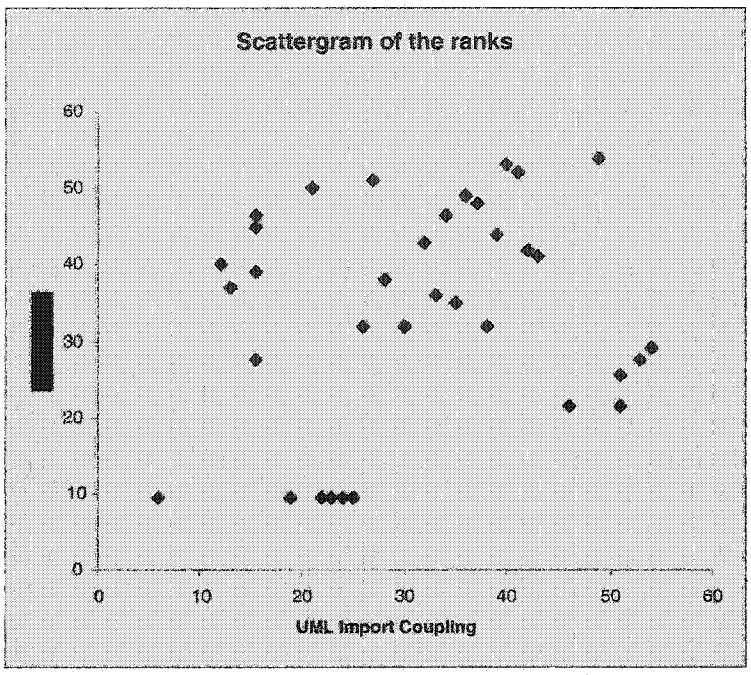

Figure 34: Spearman Rank Correlation Scatter Chart for Interactive Coupling Vs MsgSent

Table 24: Spearman Rank Correlation results for UML Interactive Coupling Vs MsgSent

Spearman Rank Correlation Coefficient	0.528
N	53
p-value	0.001
Alpha	0.05

Here we observe that correlation is not strong but better than last case because here we compare our metric with another design level metric.

Since p- value is less than the alpha-value, we conclude that the correlation is significant at the confidence level of 95 %.

As discussed before although these results show that interaction coupling metric correlates with lower strength with the source level than design level counterparts, still the source level metric we used were different coupling metric, hence better results can be expected if source level metrics were also measured in the same way as design level metrics.

7.4.4 Package Coupling

We used the two measures of Package coupling that we proposed, the External Package Coupling and Internal Package Coupling. The External Package Coupling was of type Import. We were unable to find any source level metric tool for Java that could compute package level coupling. SDMetrics [96] computes design level package coupling, both of internal and external types. We present the results obtained from these tools for the case study.

The results of Spearman Rank Correlation of Internal Package coupling metric against SDMetrics Package coupling metric are show in Table 25.

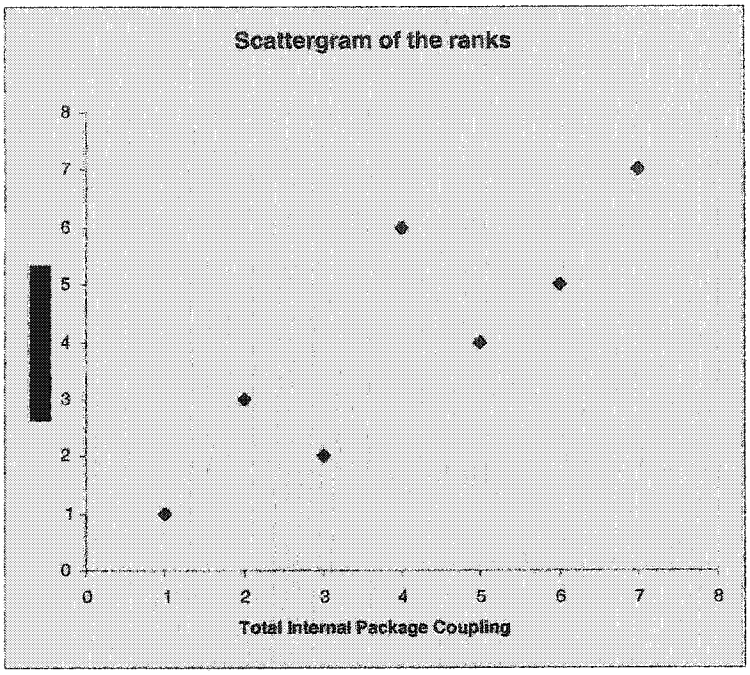

Figure 35: Spearman Rank Correlation Scatter Chart for Internal Package Coupling Vs MsgSentWithin

Table 25: Spearman Rank Correlation results for Internal Package Coupling Vs MsgSentWithin

Spearman Rank Correlation Coefficient	0.857
N	53
p-value	0.014
Alpha	0.05

Since observed p-value is less than the alpha-value, we conclude that the correlation is significant at the confidence level of 95 %.

The results of Spearman Rank Correlation of External Package coupling metric against SDMetrics Package coupling metric are shown in Table 26.

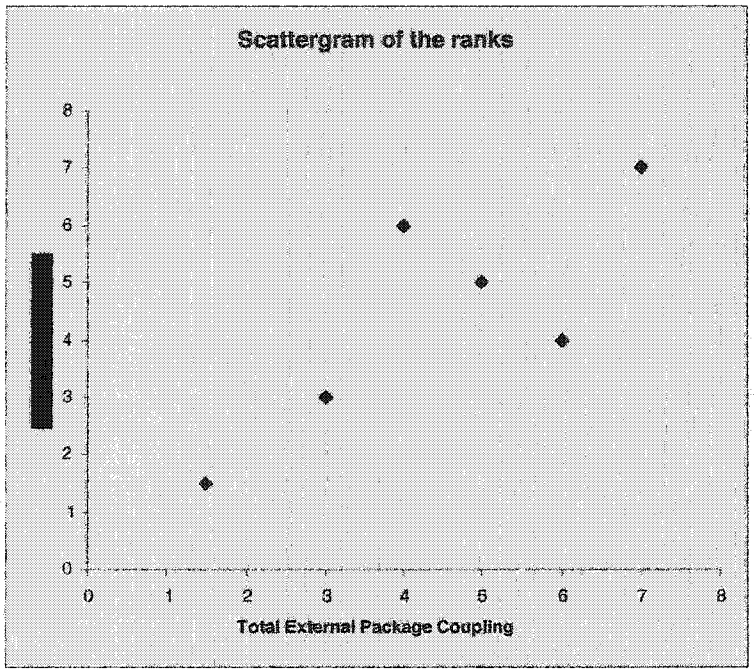

Figure 36: Spearman Rank Correlation Scatter Chart for External Package Coupling Vs MsSentOutside

Since observed p-value is less than the alpha-value as shown in Table 26, we conclude that the correlation is significant at the confidence level of 95 %.

Table 26: Spearman Rank Correlation results for Internal Package Coupling Vs MsgSentOutside

Spearman Rank Correlation Coefficient	0.855
N	53
p-value	0.014
Alpha	0.05

7.5 Results

The results of all experiments are summarized in Table 27. It can be appreciated from the results that except one all the null hypothesis were rejected, indicating correlation between UML design Metrics and source level Metrics. It can be noted here that the metrics used were still not exactly measuring the properties in same way, hence if a tool can implement source level and design metrics both in the same way, then a stronger correlation can be observed. The results for class complexity are promising because they are computed completely from design information and they are still able to predict the actual class complexity.

Table 27: Experimental Results

Metric Validated	Case Study	Source Level Counterpart	Design Level Counterpart	Results
Class Complexity	OOMeter	LOC		Significant Correlation
		WMAC1		Significant Correlation
		WMAC2		Significant Correlation
		Cyclomatic Complexity		Significant Correlation
Inheritance Coupling Metric	Java.Lang.Ref		AlMulla IC	Significant Correlation
	Java.Awt.Event		AlMulla IC	Significant Correlation
	OOMeter	Henri Li		Significant Correlation
Import Interaction Coupling Metric	OOMeter	VOD		Weak Correlation
			MsgSent	Weak Correlation
Internal Package Coupling Metric	OOMeter		MessageSent Within	Significant Correlation
External Package Coupling Metric	OOMeter		MessageSent Outside	Significant Correlation

In some case where design and code level metric differ too much, then it calls for careful investigation. For example in case study "oometer" the class complexities for class "coupling" differs significantly. Although its interface, design was good, it was ill coded because of lack of time. For the same class the values differ significantly for interactive coupling. It was because there was no coupling in design, but code contained many temporary variables coupled to other classes. This violates a design principle that coupling should be specified as much as possible at design level.

Package coupling metrics can be useful to suggest proper package composition. For example if external package coupling for a package is more than its internal package coupling, then merging the two will reduce the coupling.

Chapter 8

Conclusion

In this chapter, we present a summary of our work and suggest ways of improvement in future. The suite of UML coupling metrics proposed and implemented can prove useful in the early design stage of Object Oriented Software Development in many ways like:

- UML Inheritance metrics can reflect on the quality of inheritance hierarchy of the system. Inheritance coupling is desirable type of coupling because sub classes should be coupled with the super classes.
- UML Interaction Coupling can reflect on the complexity, maintenance efforts and error probability of the system.
- UML Component coupling can reflect on the error propagation of the modules.
- UML Interaction Coupling and Package Coupling metrics can also be used for project management. For example a class or package with high export coupling means that it provides service to large number of clients. Hence it needs the best development efforts.

- UML Interaction Coupling, UML Component Coupling and Package Coupling metrics can also be useful in refactoring. During the refactoring process they can predict how much the change in one class will affect the coupled class. Based on these different refactoring alternatives can also be compared based on development efforts.
- The difference in source and design level metrics like size and interaction coupling can indicate a need for further investigation.

8.1 Limitations and Further Work

In this section we discuss some of the directions where this work can be taken to and prove useful.

8.1.1 External Validation

The validation of design level metrics was only done with other similar metrics, but this validation if done with some external attributes like fault density, maintainability, understandability could reveal some interesting features of design level measures and help them make more useful.

8.1.2 Process Metrics

An interesting experiment could be to analyze the difference between source and design level metric measured in the same way. This can be made easy with good metric tools. The difference in measures could provide insight into the development process and help to control it.

8.1.3 Higher Level UML Metrics

The analysis of metrics possible with UML can be extended and built upon to measure other useful metrics like cohesion, reuse etc. The conceptual framework developed in Chapter 4 can be used to find those higher level metrics that can be readily measured from UML designs.

The framework can be extended to include other UML artifacts like use cases and state diagrams. The UML coupling metrics proposed are based on Package, Class and Sequence diagrams. The definition can be extended to include use cases. In a process centered on UML use cases are linked to the sequence diagrams. There are connections between use cases like inheritance, extension and inclusion. Hence the information about these connections can be added to the coupling or cohesion metrics. For example two classes implementing the same use case are cohesive functionally. Similarly if there is a dependency between two use cases then it will also show up in the classes that implement those use cases as coupling. These

metrics can then even predict the code level coupling or cohesion of the system from the early design consisting of use cases, package, class and sequence diagrams.

8.1.4 Refactoring Tool

Refactoring is an important part of software development and maintenance. Since coupling metric provides the quantitative information of connectedness of modules, it can be used to compute the total amount of change caused by a small change in a particular module. The visualization of this information can be very helpful for the designers and software architects and important addition to the use of UML in design as well as refactoring. There are some tools that provide similar facilities for portioning of VLSI circuits.

8.2 Summary and Contributions of Thesis

We proposed a suite of UML coupling metrics that can measure all dimensions of coupling from UML design artifacts (Package, Class and Sequence diagrams). We designed and implemented a UML Metric tool to extract above design features from UML design artifacts stored in XMI format and compute all the above metrics. These metrics were validated theoretically and empirically. The contributions of this thesis can be summarized as follows:

- Proposed UML Size metric
    - UML Class Size
- Proposed 6 UML Coupling metrics
    - UML Inheritance Coupling
    - UML Interaction Coupling
    - UML Component Coupling
    - Inter-Package Coupling
    - Internal Package Coupling
    - External Package Coupling
- Implemented a Metrics Tool to extract the design level information and compute above metrics from UML Models (Package, Class, Sequence diagrams) stored in XMI files.
- Theoretically and empirically validated the above metrics.

Appendix A: Tool Design

Appendix B: Experimental Results

Case Study: OOMeter

Table 28: Together class metric results for OOMeter

Class Names	Cyclomatic	WMC1	WMC2	LOC
common-Association	22	22	32	80
common-Attribute	24	24	22	80
common-ClassDeclaration	30	30	45	117
common-ClassInterface	16	16	24	62
common-DB	82	82	70	706
common-DBMetrics	23	23	28	166
common-ElementaryItem	2	2	2	7
common-InterfaceDeclaration	19	19	28	75
common-Message	28	28	42	110
common-Method	17	17	25	71
common-PackageDeclaration	13	13	19	53
common-Parameter	13	13	19	53
common-UsedAttribute	21	21	29	75
common-UsedMethod	20	20	29	74
javaparser-FileHandler	2	2	6	32
javaparser-FolderHandler	6	6	6	32
javaparser-Interpreter	237	237	119	993
javaparser-Main	16	16	14	86
javaparsertest-ElementaryMetricTest	6	6	8	53
javaparsertest-TBeingUsed		1	2	6
javaparsertest-TchildBeingUsed		3	4	13
oometer-ClassComplexity	7	7	5	69
oometer-Coupling	158	158	73	937
oometer-CTC	0	0	0	3
oometer-DIT	5	5	5	48
oometer-ICC	0	0	0	3
oometer-MeasurementTask	0	0	0	5
oometer-MeasurementTasks	15	15	24	21
oometer-MethodComplexity	5	5	5	62
oometer-Metrics	15	15	24	20
oometer-NOC	4	4	5	41
oometer-OOMeter	0	0	0	4
oometer-OOSystem	0	0	0	5
oometer-OOSystems	15	15	24	22
oometer-OSC	0	0	0	3
oometer-XMLMetric		0	0	3
oometertest-oometer	8	8	10	50
xmiparser-AssociationEndState	15	114	134	59

Class Names	Cyclomatic	WMC1	WMC2	LOC
xmiparser-AssociationState	13	15	18	43
xmiparser-AttributeState	17	13	19	58
xmiparser-ClassifierRole	14	17	18	61
xmiparser-ClassifierRoleState	15	14	21	45
xmiparser-ClassState	29	15	18	110
xmiparser-Collaboration	16	29	19	68
xmiparser-CollaborationState	17	16	24	61
xmiparser-InitialState	9	17	21	27
xmiparser-InterfaceState	17	9	18	67
xmiparser-MessageState	27	27	18	137
xmiparser-OperationState	28	28	23	82
xmiparser-PackageState	17	17	19	66
xmiparser-ParameterState	18	18	18	69
xmiparser-XMI10UML11StateTransition	50	50	34	171
xmiparser-XMIParser	114	114	134	485
xmiparsertest-PrintXMIElements	12	12	3	147

Table 29: XMI Parser Class size results for OOMeter

Class Names	UML Class Size	Relative Values to Sum	Relative Values to Maximum
common-Association	654	0.005161555	0.012716313
common-Attribute	395	0.003117453	0.007680342
common-ClassDeclaration	831	0.00655849	0.016157885
common-ClassInterface	408	0.003220053	0.007933113
common-DB	51430	0.405900273	1
common-DBMetrics	840	0.00662952	0.01633288
common-ElementaryItem	44	0.000347261	0.000855532
common-InterfaceDeclaration	512	0.00404085	0.009955279
common-Message	825	0.006511136	0.016041221
common-Method	401	0.003164807	0.007797006
common-PackageDeclaration	323	0.002549208	0.006280381
common-Parameter	380	0.002999069	0.007388684
common-UsedAttribute	590	0.004656449	0.011471904
common-UsedMethod	590	0.004656449	0.011471904
javaparser-FileHandler	163	0.001286443	0.003169356
javaparser-FolderHandler	263	0.002075671	0.005113747
javaparser-Interpreter	6476	0.051110445	0.125918724
javaparser-Main	1316	0.010386248	0.025588178
javaparsertest-ElementaryMetricTest	167	0.001318012	0.003247132
javaparsertest-TBeingUsed	41	0.000323584	0.0007972
javaparsertest-TChildBeingUsed	126	0.000994428	0.002449932
oometer-ClassComplexity	191	0.001507427	0.003713786

Class Names	UML Class Size	Relative Values to Sum	Relative Values to Maximum
oometer-Coupling	2402	0.018957271	0.046704258
oometer-CTC	0	0	0
oometer-DIT	135	0.001065459	0.002624927
oometer-ICC	0	0	0
oometer-MeasurementTask	40	0.000315691	0.000777756
oometer-MeasurementTasks	293	0.00231244	0.005697064
oometer-MethodComplexity	183	0.001444288	0.003558234
oometer-Metrics	293	0.00231244	0.005697064
oometer-NOC	135	0.001065459	0.002624927
oometer-OOMeter	20	0.000157846	0.000388878
oometer-OOSystem	20	0.000157846	0.000388878
oometer-OOSystems	293	0.00231244	0.005697064
oometer-OSC	0	0	0
oometer-XMLMetric	0	0	0
oometertest-oometer	261	0.002059887	0.005074859
xmiparser-AssociationEndState	4985	0.039343046	0.096927863
xmiparser-AssociationState	1213	0.009573343	0.023585456
xmiparser-AttributeState	3357	0.026494404	0.065273187
xmiparser-ClassifierRole	384	0.003030638	0.007466459
xmiparser-ClassifierRoleState	2030	0.016021341	0.039471126
xmiparser-ClassState	5982	0.047211655	0.116313436
xmiparser-Collaboration	447	0.003527852	0.008691425
xmiparser-CollaborationState	1583	0.012493489	0.030779701
xmiparser-InitialState	701	0.005532493	0.013630177
xmiparser-InterfaceState	3038	0.023976765	0.059070581
xmiparser-MessageState	7680	0.060612757	0.149329185
xmiparser-OperationState	3089	0.024379272	0.06006222
xmiparser-PackageState	3202	0.0252711	0.062259382
xmiparser-ParameterState	4484	0.035389011	0.087186467
Xmiparser-XMI10UML11StateTransition	5597	0.044173125	0.108827533
xmiparser-XMIParser	6854	0.054093729	0.13326852
Xmiparsertest-PrintXMIElements	486	0.003835651	0.009449738

Table 30: Case Study 2 Elish Results

Class	Inheritance Coupling
java.awt.event.ActionEvent	0.7118
java.awt.event.AdjustmentEvent	0.6676
java.awt.event.ComponentAdapter	0.7447
java.awt.event.ComponentEvent	0.8107
java.awt.event.ContainerAdapter	0.8274
java.awt.event.ContainerEvent	0.7996
java.awt.event.FocusAdapter	0.8274

Class	Inheritance Coupling
java.awt.event.FocusEvent	0.8469
java.awt.event.InputEvent	0.8222
java.awt.event.InputMethodEvent	0.7677
java.awt.event.InvocationEvent	0.7165
java.awt.event.ItemEvent	0.7362
java.awt.event.KeyAdapter	0.7839
java.awt.event.KeyEvent	0.2029
java.awt.event.MouseAdapter	0.7092
java.awt.event.MouseEvent	0.7047
java.awt.event.MouseMotionAdapter	0.8274
java.awt.event.PaintEvent	0.7996
java.awt.event.TextEvent	0.8738
java.awt.event.WindowAdapter	0.6476

Table 31: Case Study 2 Henri Li Metric

Class	Inheritance Coupling
java.awt.event.ActionEvent	1
java.awt.event.AdjustmentEvent	1
java.awt.event.ComponentAdapter	1
java.awt.event.ComponentEvent	8
java.awt.event.ContainerAdapter	1
java.awt.event.ContainerEvent	2
java.awt.event.FocusAdapter	1
java.awt.event.FocusEvent	2
java.awt.event.InputEvent	4
java.awt.event.InputMethodEvent	1
java.awt.event.InvocationEvent	1
java.awt.event.ItemEvent	1
java.awt.event.KeyAdapter	1
java.awt.event.KeyEvent	3
java.awt.event.MouseAdapter	1
java.awt.event.MouseEvent	3
java.awt.event.MouseMotionAdapter	1
java.awt.event.PaintEvent	1
java.awt.event.TextEvent	2
java.awt.event.WindowAdapter	1

Table 32: Interaction Coupling Results

Class	VOD	CF	CDBC	Dep Client	Dep Supp	Msg Sent	UML Interaction Coupling
common-Association	1					1	0.052107177
common-Attribute	1					2	0.099008124

Class	VOD	CF	CDBC	Dep Client	Dep Supp	Msg Sent	UML Interaction Coupling
common-ClassDeclaration	1					1	0.098626633
common-ClassInterface	1					0	0.002356268
common-DB	8					1995	0.957115928
Common-DBMetrics	6					0	0.000493694
Common-ElementaryItem	0					1	0.047596607
Common-InterfaceDeclaration	1					1	0.050895382
Common-Message	1					1	0.051030026
Common-Method	1					3	0.147479916
Common-PackageDeclaration	1					1	0.049952875
Common-Parameter	1					2	0.098649073
Common-UsedAttribute	1					0	0.009784121
Common-UsedMethod	1					5	0.24931556
javaparser-FileHandler	2					6	0.00139132
javaparser-FolderHandler	0					8	0.001929895
javaparser-Interpreter	3					176	0.004802298
javaparser-Main	2					30	0.008662089
javaparsertest-ElementaryMetricTest	3					6	0.001840133
javaparsertest-TBeingUsed	0					0	0
javaparsertest-TChildBeingUsed	1					0	0.000179525
oometer-ClassComplexity	2					10	0.000493694
oometer-Coupling	9					35	0.004106638
oometer-CTC	0					0	0
oometer-DIT	3					6	0.000314169
oometer-ICC	0					0	0
oometer-MeasurementTask	0					0	0
oometer-MeasurementTasks	0					0	0
oometer-MethodComplexity	2					6	0.000314169
oometer-Metrics	0					0	0
oometer-NOC	2					6	0.000314169
oometer-OOMeter	0					0	0
oometer-OOSystem	0					0	0
oometer-OOSystems	0					0	0
oometer-OSC	0					0	0
oometer-XMLMetric	0					0	0
oometertest-oometer	3					0	0
xmiparser-AssociationEndState	3					46	0.002647996
xmiparser-AssociationState	3					14	0.003096809
xmiparser-AttributeState	5					36	0.001997217
xmiparser-ClassifierRole	1					0	0.000201966
xmiparser-ClassifierRoleState	1					22	0.002468471
xmiparser-ClassState	6					73	0.005251111
xmiparser-Collaboration	1					0	0.000179525

Class	VOD	CF	CDBC	Dep Client	Dep Supp	Msg Sent	UML Interaction Coupling
xmiparser-CollaborationState	1					19	0.002199183
xmiparser-InitialState	1					3	0.001077151
xmiparser-InterfaceState	3					38	0.001705489
xmiparser-MessageState	5					91	0.003164131
xmiparser-OperationState	5					40	0.00381491
xmiparser-PackageState	3					40	0.003186572
xmiparser-ParameterState	5					48	0.002715318
Xmiparser-XMI10UML11StateTransition	1					37	0.003276334
xmiparser-XMIParser	2					18	0.019096988
Xmiparsertest-PrintXMIElements	3					324	0.005161348

Table 33: Package Coupling Results

Package	MsgSend Outside	MsgSend Within	External Package Coupling	Internal Package Coupling
Oometer	63	180	0.003678649	0.008611783
xmiparsertest	1	3	0.001531281	5.85371E-06
xmiparser	122	430	0.002876326	0.00654835
javaparsertest	2	9	0.00398246	0.000288783
javaparser	194	80	0.01951537	0.008161047
Common	0	2278	0	0.951881529
Oometertest	0	12	0	7.51226E-05

Bibliography

[1] Oxford English Dictionary, Oxford University Press, 2000

[2] J. Resnick and D. Halliday. Fundamentals of Physics. Chapter 5, John Wiley and Sons, New York, third edition, 1988.

[3] Tony R. Kuphaldt, "Lessons In Electric Circuits" Design Science License, 2001

[4] Robert A. Capobianco, "Design Considerations for Optical Coupling of Flashlamps and Fiber Optics" Retrieved December 06, 2003 from the World Wide Web: http://optoelectronics.perkinelmer.com/content/whitepapers/OpticalCoupling.pdf

[5] Francisca Losavio, Ledis Chirinos, Nicole Lévy, Amar Ramdane-Cherif, "Quality Characteristics for Software Architecture", Journal Of Object Technology, Vol. 2, No. 2, March-April 2003

[6] Francisca Losavio, "Quality Models to Design Software Architecture" Journal Of Object Technology, Vol. 1, no. 4, September-October 2002

[7] Peyman Oreizy, Nenad Medvidovic, Richard N. Taylor David S., Rosenblum, "Software Architecture and Component Technologies: Bridging the Gap", Workshop on Compositional Software Architectures, 1997

[8] Mary Shaw, "The Coming-of-Age of Software Architecture Research", International Conference on Software Engineering, 2001

[9] Daniel Starr, "What's Holding Up the Roof? Site, structure, stuff ... and software: In creating programs and buildings, the elements of architecture apply."Software Development July 2003

[10] Mike Moore,Rick Kazman,Mark Klein,Jai Asundi,Quantifying the Value of Architecture Design Decisions: Lessons from the Field"

[11] David Miranda, Mario Piattini, Marcela Genero, "Defining and Validating Metrics for UML Statechart Diagrams", 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)

[12] Hyoseob Kim, Cornelia Boldyreff, "Developing Software Metrics Applicable to UML Models", 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)

[13] Ed Seidewitz, "What do models mean?" White paper, InteliData Technologies Corporation, March 2003

[14] Jack W. Reeves, "What is Software Design?" C++ Journal, 1992

[15] Subrata Dasgupta"Design theory and computer Science", Cambridge tracts in Theoretical Computer Science, Cambridge University Press, 1991

[16] Boehm, B."Software LifeCycle Factors", in Handbook of Software Engineering, CR Vick and CV Ramamoorthy, Van Nostrand-Reinhold, 494-518, 1984

[17] Zelkowitz, M.V., Shaw, A.C. and Gannon, J.D. "Principles of Software Engineering and Design", Prentice-Hall, 1979

[18] OMG (Object Management Group) Specification, UML (Unified Modeling Language) Retrieved December 06, 2003 from the World Wide Web: http://www.omg.org/uml/.

[19] World Wide Web Consortium (W3C) Specification, XML (Extensible Markup Language), Retrieved December 06, 2003 from the World Wide Web: http://www.w3.org/XML/.

[20] OMG (Object Management Group) Specification, XMI (XML Metadata Interchange) Retrieved December 06, 2003 from the World Wide Web: http://www.omg.org/technology/documents/formal/xmi.htm.

[21] OMG (Object Management Group) Specification, MOF (Meta-Object Facility) Retrieved December 06, 2003 from the World Wide Web: http://www.omg.org/technology/documents/formal/mof.htm.

[22] OMG (Object Management Group) Specification, MDA (Model Driven Architecture) Retrieved December 06, 2003 from the World Wide Web: http://www.omg.org/mda.

[23] World Wide Web Consortium (W3C) Specification, XSLT (XSL Transformations), Retrieved December 06, 2003 from the World Wide Web: http://www.w3.org/XML/.

[24] Fazli Canand, Esen A. Ozcarahan, "Concepts and Effectivveness of the Cover-Coefficient-Based Clustering Methodology for Text Databases", ACM Transactions on Database Systems, Vol 15, No 4, Dec 1990, pp483-517.

[25] Jarallah Al-Ghamdi, M. Shafique, S. Al-Nasser, T. Al-Zubaidi, "Measuring the Coupling of Procedural Programs", ACS/IEEE International Conference on Computer Systems and Applications 2001.

[26] Jarallah AlGhamdi, Programming Languages: A Quantitative Methodology for Assessments Software Metrics to Measure Syntactic Properties, Ph.D. Thesis, Arizona State University, Aug. 1994.

[27] Elish Mahmoud Omar, "Measuring inheritance coupling in object-oriented systems", KING FAHD UNIVERSITY OF PETROLEUM

AND MINERALS (SAUDI ARABIA), 138 pages, AAT 1398019, 1999.

[28] Al-Mulla, MS Thesis, "Measuring class coupling in object-oriented design", KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS (SAUDI ARABIA), 181 pages, AAT 1390780, 1998.

[29] J-Y Chen and J-F Lu, "A New Metric For Object-Oriented Design", Information and Software Technology, pp 232- 240, Jun. 1993.

[30] D.L. Parnas "On the criteria to be used in decomposing systems into modules" Communications of the ACM, Volume 15 No. 12, 1972

[31] W.P. Stevens, G.J. Myers, L.L. Constantine "Structured design" IBM Systems Journal Volume 13 No. 2, 1974.

[32] G. Myers, "Composite/Structured Design", Van Nostrand Reinhold, 1978.

[33] P. Coad, E. Yourdon, "Object-Oriented Design", Prentice Hall (Yourdon Press), 1991

[34] Edward V. Berard, "Essays on Object-Oriented Software Engineering", volume 1. Prentice Hall, 1993.

[35] M. Xenos, D. Stavrinoudis, K. Zikouli, D. Christodoulakis "Object-Oriented Metrics – a survey" Proceedings of the FESMA 2000, Federation of European Software Measurement Associations, Madrid, Spain, 2000.

[36] S.R. Chidamber, C.F. Kemerer, "Towards a Metrics Suite for Object Oriented design", in A. Paepcke,(ed.) Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA'91), October 1991. Published in SIGPLAN Notices, 26 (11), 197-211, 1991.

[37] S.R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, 20 (6), 476-493, 1994.

[38] V. R. Basili, L. Briand, and W. Melo, "A Validation of OO Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, 1996.

[39] S. R. Chidamber and C. F. Kemerer, "A metric suite for object oriented design," IEEE Transactions on Software Engineering, pp. 467–493, 1994.

[40] N.I. Churcher, M.J. Shepperd, "Comments on 'A Metrics Suite for Object-Oriented Design'", IEEE Transactions on Software Engineering, 21 (3), 263-265, 1995.

[41] N.I. Churcher, M.J. Shepperd, "Towards a Conceptual Framework for Object Oriented Software Metrics", Software Engineering Notes, 20 (2), 69-76, 1995.

[42] Wild, F. III., "Managing Class Coupling: Applying the Principles of Structured Design to Object Oriented Programming", Unix Review 9, no. 10, October 1991.

[43] J. Eder, G. Kappel, M. Schrefl, "Coupling and Cohesion in Object-Oriented Systems", Conference on Information and Knowledge Management, Baltimore, USA, 1992.

[44] M. Hitz, B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented systems", in Proc. Int. Symposium on Applied Corporate Computing, Monterrey, Mexico, October 1995.

[45] M. Hitz, B. Montazeri, "Chidamber & Kemerer's Metrics Suite: A Measurement Theory Perspective", IEEE Transactions on Software Engineering, 22 (4), 276-270, 1996.

[46] Lionel C. Briand, John W. Daly, Jürgen Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", IEEE Transactions on Software Engineering, 1996.

[47] L. Briand, P. Devanbu, W. Melo, "An Investigation into Coupling Measures for C++", International Conference on Software Engineering, 1997.

[48] D. Troy and S. Zweben, "Measuring the quality of structured designs," J. Systems and Software, vol. 2, pp. 113-120, 1981.

[49] Brain Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity, Prentice Hall PTR, 1996.

[50] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, vol. 23, no. 2, pp. 111-122, 1993.

[51] M. Lorenz, Object-Oriented Software Development: A Practical Guide, Prentice Hall, NJ, 1993.

[52] M. Lorenz and J. Kidd, Object-Oriented Software Metrics, Prentice Hall, NJ, 1994.

[53] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL - A query language for XML. In Proc. of the Int. World Wide Web Conference, Canada, 1999.

[54] Serge Abiteboul, Jennifer Widom, Tirthankar Lahiri "A Unified Approach for Querying Structured Data and XML" QL'98 - Query Languages 1998

[55] Gustavo Arocena , Alberto Mendelzon , George Mihaila "Query Languages for the Web" QL'98 - Query Languages 1998

[56] David Garlan and Mary Shaw, "An introduction to software architecture," In Advances in Software Engineering and Knowledge Engineering," Volume 1, World Scientific Publishing Company, 1993.

[57] Paul Adamczyk, "The Anthology of the Finite State Machine Design Patterns", The 10th Conference on Pattern Languages of Programs 2003

[58] Michele Lanza, St´ephane Ducasse, "Beyond Language Independent Object Oriented Metrics: Model Independent Metrics", QAOOSE 2002 Proceedings (6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering), pp. 77 - 84, 2002.

[59] Erich Schikuta, "Dynamic Software Metrices", 1993

[60] Paques H,L Delcambre, "A Mechanism for Assessing Class Interactions Using Dynamic Coupling During the Analysis Phase", XVIII Brazilian Symposium on Software Engineering - SBES', 1999

[61] Tarja Systä, "Understanding the Behavior of Java Programs", Working Conference on Reverse Engineering, 2000

[62] Raitalaakso T, "Dynamic Visualization of C++ Programs with UML Sequence Diagrams", Master of Science Thesis, December 2000. Retrieved December 06, 2003 from the World Wide Web: http://practise.cs.tut.fi/pub/papers/rafu_master.pdf

[63] J2U Sequence Diagram Reverse Engineering Tool, Retrieved December 06, 2003 from the World Wide Web: http://www.nasra.fr/flash/NASRA.html

[64] Sherif Yacoub, Tom Robinson, and Hany H. Ammar, "Dynamic Metrics for Object Oriented Designs"", In Proceedings of the Sixth International Symposium on Software Metrics, Metrics'99, Boca Raton, Florida USA, November 4-6 1999, pp50-61.

[65] Aine Mitchell, "DYNAMIC COUPLING AND COHESION METRICS FOR JAVA PROGRAMS",Department of Computer Science, NUI Maynooth, MASTERS THESIS 2002.

[66] Alaa Ibrahim, Sherif M. Yacoub, Hany H. Ammar, "Architectural-Level Risk Analysis for UML Dynamic Specifications," Proceedings of the 9th International Conference on Software Quality Management (SQM2001), Loughborough University, England, April 18-20, 2001, pp. 179-190

[67] Ahmed Hassan, Walid M. Abdelmoez, Rania M. Elnaggar, Hany H. Ammar, "An Approach to Measure the Quality of Software Designs from UML Specifications," 5th World Multi-Conference on Systems, Cybernetics and Informatics and the 7th international conference on information systems, analysis and synthesis ISAS July. 2001.

[68] Erik Arisholm, Dynamic Coupling Measures for Object-Oriented Software, 8th International Symposium on Software Metrics, 4-7 June

2002, Ottawa, Ont., Canada, pp. 33-42, IEEE Computer Society, 2002.

[69] Erik Arisholm, Lionel Briand and Audun Føyen, Dynamic Coupling Measurement for Object-Oriented Software, Submitted to IEEE Transactions on Software Engineering, 2003

[70] Mark Shereshevsky, Habib Ammari, Nicholay Gradetsky, Ali Mili and Hany H. Ammar, "Information Theoretic Metrics for Software Architectures," International Computer Software and Applications Conference (COMPSAC 2001), IEEE Computer Society, Chicago, IL., October 2001

[71] Massimo Carbone, Giuseppe Santucci, "Fast and Serious: a UML based metric for effort estimation", 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)

[72] Aline Lúcia,Fernando Brito, "Formalizing Object-Oriented Design Metrics upon the UML Meta-Model", 16th Brazilian Symposium on Software Engineering, Gramado, Brazil, 2002

[73] Aline Lúcia Baroni, Fernando Brito e Abreu, "An OCL-Based Formalization of the MOOSE Metric Suite", 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2003)

[74] Fernando Brito, Rogério Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", 4th Int. Conf. on Software Quality", McLean, VA, USA, 3-5 October 1994

[75] Marcela Genero, Mario Piattini, "Empirical validation of measures for class diagram structural complexity through controlled experiments",5th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2001)

[76] Scott A. Whitmire, Object-Oriented Design Measurement, John Wiley & Sons, 1997

[77] XPB4J-0.90, XML Processing Benchmark for Java (XPB4J)(Accessed 2002-07-01 http://www.pankaj-k.net/xpb4j/)

[78] Trevor Paterson, Object-Oriented Software Design Metrics from XMI, MSc Project Dissertation for IT (Systems), 2002

[79] Peter In, SangEun Kim,Matthew R. Barry, "UML-based Object-Oriented Metrics for Architecture Complexity Analysis", Ground System Architectures Workshop (GSAW) 2003

[80] Anh Le, Fang Lu, Yujun Quin and Qing Zhang, "UML Metric Tool," Final Report, University of Houston at Clear Lake, December 2001.

[81] Jukka Paakki, Anssi Karhinen, Juha Gustafsson, Lilli Nenonen and A. Inkeri Verkamo, "Software metrics by architectural pattern mining," Proceedings of the International Conference on Software: Theory and Practice (16th IFIP World Computer Congress), 325-332, Beijing, China, August 2000.

[82] Reibing R. The impact of Pattern Use on Design Quality. OOPSLA 2001 – Workshop

[83] Marcela Genero, Mª Esperanza Manso, Mario Piattini, Francisco García "Early metrics for object oriented information systems"

[84] Amador Durán Antonio Ruiz Miguel Toro "Implementing Automatic Quality Massimo Carbone and Giuseppe Santucci "Fast&&Serious: a UML based metric for effort estimation"

[85] H. Diab F. Koukane M. Frappier R. St-Denis "μcROSE : Functional Size Measurement for Rational Rose RealTime" 2002

[86] Marcela Genero, David Miranda and Mario Piattini "Defining and Validating Metrics for UML Statechart Diagrams"

[87] Hyoseob Kim, Cornelia Boldyreff "Developing Software Metrics Applicable to UML Models" 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)

[88] A. Minkiewicz. "Measuring Object-Oriented Software with Predictive Object Points". ASM'97, Atlanta, October 1997.

[89] G. Karner. "Resource Estimation for Objectory Projects". Objectory Systems, 1993.

[90] Whitmire, Scott. "3-D Function Points: Applications for Object-Orented Software in Applications of Software Management Conference (1996)

[91] H. Sneed. Estimating the Costs of Object-Oriented Software. Proceedings of Software Cost Estimation Seminar, System Engineering Ltd. , Durham, UK, March 1995.

[92] S. Sarferaz, W. Hesse. "CEOS-A Cost Estimation Method for Evolutionary, Object-Oriented Software Development". Proceedings of New Approaches in Software Measurement, 10th International Workshop, IWSM 2000, Berlin, Germany, October 2000.

[93] Rufai, Raimi Ayinde, "New structural similarity metrics for UML models", MS Thesis, KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS (SAUDI ARABIA), 128 pages, AAT 1413039, 2003. Retrieved December 06, 2003 from the World Wide Web: http://www.ccse.kfupm.edu.sa/~rrufai/thesis/thesis_rrufai_v1.0.1.pdf

[94] Together Case Tool, Together Inc., Retrieved December 06, 2003 from the World Wide Web: http://www.together.com/.

[95] Ration Rose Case Tool, IBM Inc., Retrieved December 06, 2003 from the World Wide Web: http://www.rational.com/.

[96] SDMetrics UML Metric Tool, SDMetrics Inc., Retrieved December 06, 2003 from the World Wide Web: http://www.sdmetrics.com/.

[97] Protégé-2000 is an ontology editor, Retrieved December 06, 2003 from the World Wide Web: http://protege.stanford.edu/

[98] Booch, G., J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, MA: Addison-Wesley, 1998.

[99] El-Emam, K. A Methodology for Validating Software Product Metrics. National Research Council of Canada, NRC/ERB 1076, June 2000.

[100] Fenton, N. Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, 20(3), Mar. 1994.

[101] Fenton, N.E. and S.L. Pfleeger. *Software Metrics: A Rigorous & Practical Approach.* PWS Publishing Co. 1997.

[102] O. J. Dahl, W. Dijkstra and C. A. R. Hoare, Structured Programming, Academic Press, N. Y., 1972.

[103] Gamma, E., R. Helm , R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[104] Grand, M. *Patterns in Java, Volume I: A Catalog of Reusable Design Patterns*. John Wiley & Sons, 1998.

[105] Kobryn, C. UML 2001: A Standardization Odyssey. *Communications of the ACM*, Vol. 42, No. 10, October, 1999, pp. 29-37.

[106] Kruchten, P.B. The 4+1 View Model of Architecture. *IEEE Software*, Nov. 1995, pp. 42-50.

[107] Miller, J. Replicating software engineering experiments: a poisoned chalice or the Holy Grail. [Draft Book Chapter]. Oct. 2000. Available at http://sern.ucalgary.ca/courses/SENG/693/F00/readings/JamesMiller.pdf [Accessed 2002-12-10].

[108] Morasca, S., L.C. Briand, V.R. Basili, E.J Weyuker and M.V. Zelkowitz. Comments on "Towards a Framework for Software Measurement Validation." *IEEE Transactions on Software Engineering*, 23(3), Mar. 1997.

[109] Oestereich, B. *Developing Software with UML- Object-Oriented Analysis and Design in Practice (2nd Ed.)*. Addison-Wesley, 2001.

[110] OMG. *Introduction to OMG's Unified Modeling Language.* http://www.omg.org/gettingstarted/what_is_uml.htm (Accessed 2002-07-01).

[111] Rich, C. and R. C. Waters. "The Programmer's Apprentice: A Research Overview", *IEEE Computer*, vol. 21, no. 11, Nov. 1988, pp. 11-25.

[112] Schneiderwind, N.F. Methodology for Validating Software Metrics. *IEEE Transactions on Software Engineering*, 18(5), Mar. 1992.

[113] S. Siegel, N.J. Castellan, Nonparametric Statistics for the Behavioural Sciences (2nd edn), Pg 284, McGraw-Hill, New York, 1988.

[114] E.J. Weyuker, "Evaluating Software Complexity Measures," IEEE Trans. Software Eng., vol. 14, no. 9, pp. 1357-1365, Sept. 1988.

[115] XLSTAT Excel Add-on for statistics and data analysis (Accessed 2002-07-01 http://www.xlstat.com/

Index

Vita

Sohel Khan is a Graduate (M.S) Student and Research Assistant at Information and Computer Science, King Fahd University of Petroleum and Minerals. He did Bachelor of Engineering (B.E) in Electronics Design Technology from Nagpur University, India. In June 1999 he joined Novatech Software, STP, Nagpur as a Software Engineer. In February 2000 he joined Icode Inc, Bangalore. He was part of the team that designed & coded Everest, a product that won Microsoft Small Business Solution of the Year award at Asia Fusion 2001. He has worked in wide areas of system development from Database utilities to ECommerce solutions using OOAD, J2EE, DCOM, VC++, Delphi and Oracle. His Research interests are software metrics, process improvement, MDA, grid computing and protocol engineering. He is a member of GGF (Global Grid Forum), JEDI (Joint Endeavor of Delphi Innovators) Project and 1175 IEEE Working Group "Recommended Practice for CASE Tool Interconnection". He can be reached at sohelmk@yahoo.com or sohel@ccse.kfupm.edu.sa.