Toward Refactoring of DMARF and GIPSY Case Studies – a Team 3 SOEN6471-S14 Project Report

Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet Singh, Mukesh Kumar, Saleh Ahmed, and Shivam Patel

Department of Computer Science and Software Engineering

Concordia University, Montreal, Quebec, Canada

{ *a_dewal ,as_arora,kan_si,lo_sing,mu_kumar,sale_ahm,shi_pat*}*@encs.concordia.ca*

***Abstract*—The software architecture of a system is an illustration of the system which supports the understanding of the behaviour of the system. The architecture aids as the blueprint of the system, defining the work obligations which must be conceded by design and implementation teams. It is an artifact for early enquiry to make sure that a design methodology will produce a standard system. This paper depicts the software architecture and design of two frameworks DMARF and GIPSY. Primarily it inaugurates a comprehensive understanding of the frameworks and their applications. DMARF is high-volume processing of recorded audio, textual, or imagery data for pattern recognition and biometric forensic analysis, whereas GIPSY system provides a platform for a distributed multi-tier demand driven evaluation of heterogeneous programs. Secondly, the paper illustrates the use of several tools for the code analysis for both platforms and provides the outcome of the analysis. Thirdly, it establishes the architecture and design of the systems. Fourthly, it fuses the architecture for both the systems into one. The paper ends with depicting properties like code smells and refactoring to improve code quality for the frameworks.**

***Keywords:* Distributed Modular Audio Research Framework (DMARF), General Intentional Programming System (GIPSY), Software Architecture, Refactoring, Intentional Programming, Code Smells, Software Design, Code Analysis**

I. Introduction

In this revolutionary era of technology, the expedition of discoveries is orchestrated many new software and hardware frameworks which created the problem of complexity. The software and hardware complexity imperils the system's life which is a serious issue. Much inventiveness has augmented towards complexity reduction in both software and hardware with introduction of new concepts and hypotheses. So the two open source platforms are also the part of this concept; one is DMARF (Distributed Modular Audio Recognition Framework) and other is GIPSY (General Intensional Programming System). DMARF is a biologically stimulated system which is constructed on pattern recognition, signal processing and natural language processing(NLP) while GIPSY is on Intensional programming which is a generalization of unidimensional contextual programming such as temporal programming. Intensional programming is also known as multidimensional programming because the expressions involved are acceptable to differ in an arbitrary numbers of dimensions; the context of assessment is a multidimensional context.

II. Background

A. OSS Case Studies

1) DMARF: The base of DMARF is Classical Modular Audio Recognition Framework (MARF). The pipeline stages of MARF were turned into distributed nodes to make it DMARF. Classical MARF is an open source research platform where pattern recognition, NLP and signal processing algorithms are assembled and written in Java. They were organized by keeping modularity and extensibility in mind so that additional algorithms and experiments can be facilitated [2].

The pipeline stages are the mainstay of MARF; the stages communicate to acquire the data they require in chainlike way. Figure 1 presents the pattern recognition pipeline of MARF. The pipeline comprises of four rudimentary phases which are sample loading, preprocessing, feature extraction and classification/training. This version of MARF was protracted in such a way that the pipeline stages can be run as distributed nodes [4].

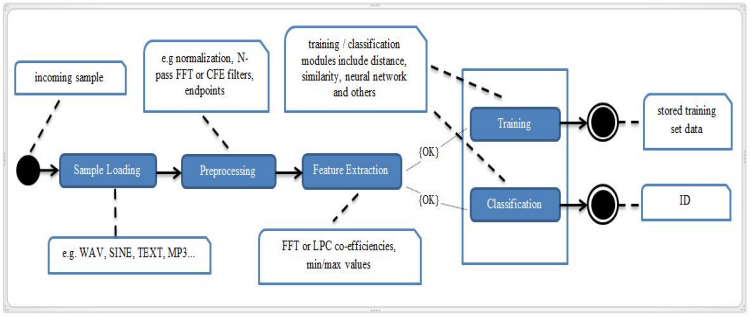

Figure 1. MARF's Pattern Recognition Pipeline [4, p.418]

Figure 2 illustrates the distributed version of the pipeline. The frontend are the modules for the client interaction which in-turn invokes services in Back-end level in a pipeline mode. Backend contains services like: primary-backup replication,

monitoring, and disaster recovery modules through delegates [2].

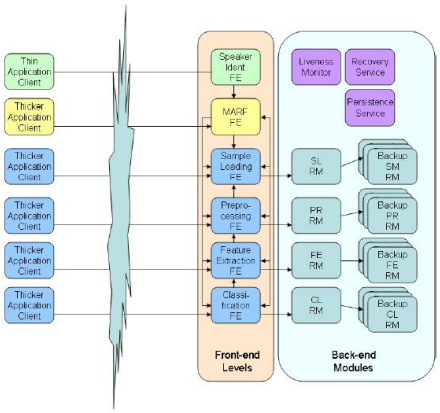

Figure 2. DMARF Pipeline [2, p. 12]

MARF which is made distributed by DMARF can be used for high volume processing of recorded audio, imagery or textual data for biometric forensic analysis and pattern recognition. Maximum weight of the applicability is in audio, for instance conference recordings. Again, for forensic analysis and subject identification and classification, DMARF can be used in a police department for processing recorded phone conversations [4].

For DMARF the key architectural principles are: platform independency, database independency API, communication technology independency, reasonable efficiency, simplicity, maintainability, architectural consistency and separation of concern [2].

The architecture of DMARF system is layered. The top level has front-end as well as back-end. The front-end occurs both on client and server side. The client side can be text-interactive, or non-interactive, or a web form/servlet. The MARF pipeline, the application-specific front-end and pipeline stage services exist on the front-end on the server side. The pipeline stages are tangled not only to the database and other storage management but also a back-end for the client connecting in [4].

For DMARF execution, naming and implementation repository service needs to run on the server side. And for Web services there should be both DNS running and a web servlet container. DMARF uses Apache Tomcat servlet container. Again, Java Runtime Environment (JRE), a servlet container environment and a browser to view/submit a web form are required on the web services client side [2].

As per as the communication paths concern, all modules communicate through message passing between methods. Additionally for Web Services, JAX-RPC (Java XML Remote Procedure Call) built operation over SOAP (Simple Object Access Protocol) is used. A WAL (writeahead message-logging) protocol is designed for DMARF for the disaster recovery and it also permits to be stretched for backup replication and point-in-time recovery (PITR). It is also extended by a "warm standby" add-on which would jump in if the primary server fails [4].

Research has been done to make the security features more efficient for DMARF. There is hazard of data alteration, incite malicious code injection and data poisoning for the node. Mainly data integrity, data availability, confidentiality and data authentication is considered to make the system more secure. Some of the security risk like maliciously induced incorrect computation results and cache poisoning can be solved through security optioning in SNMPv3, SSL and SSH. Generalised proposed solutions are Java Data Security Framework (JDSF), proxy certificates and implementing the security layer in those systems [3].

There is no threat of malicious code injection in DMARF because DMARF does not have any concern with explicit code execution. JDSF (JAVA Data Security Framework) is designed to support various characteristics of data security like: SQL randomization, Authentication, data confidentiality and data integrity. Threat model is proposed for the issues of security. The model deals mainly with confidentiality, integrity, availability and authentication. JDSF is installed on the computing and generating node and it is invoked when data enter and leaves the system [3].

With the advancement in IT industry, maintaining complex distributed system becomes the greatest challenge. Autonomic computing can be viewed as a solution, which moderates the maintenance workload of complex system, by transforming complex systems into self-managing autonomous systems. There are different functional areas in autonomic computing which are self-configuration, self-healing, self-optimization and self-protection [5].

DMARF's four stages of pipeline have to perform heavy computations, matrix operations and I/O-bound data processing. Moreover, each stage runs independently in distributed environment, so complex system's workload can be reduced by implementing self-managing or self-optimizing autonomic systems [11].

ASSL is used to implement self-optimizing autonomic policy. There are two major functional requirements that should be accomplished to apply self-optimization property on DMARF which are as follows:

- Training set classification data replication: DMARF systems perform a lot of data processing at its pipeline. Computation Data will be stored at different hosts that run classification service, which results in re-computation of already computed data on other classification hosts. To prevent this, classification stage nodes should communicate to exchange data they have acquired which helps in optimizing data computation/redundancy.
- Selection of communication protocol dynamically: DMARF using self-optimization policy that should be able to select readily available and most efficient communication protocol at all of its nodes [11].

A self-protection mechanism is incorporated into DMARF using ASSL (Autonomic System Specific Language). Special autonomic manager (AM) is added to each DMARF stage

that converts each stage into AE, which constitutes autonomic DMARF, being capable of doing self-management [5].

ASSL is used to generate/create architectural model for autonomic systems. ASSL gave a scalable multi-tier specification model that describes the configuration of infrastructure elements required by Autonomic Systems (AS). As shown in Figure 3, there are three major tiers of the model [5].

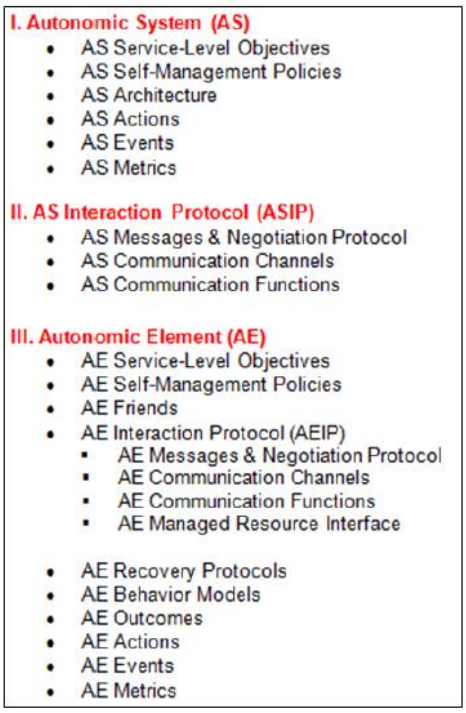

Figure 3. ASSL Multi-Tier Model [5, p. 178]

The concept of self-forensics and the idea of its implementation in ASSL are described through their finding core works. These preliminary findings and discussions are currently at conceptual level, but it is believed that a concrete formal model can be formed [12].

Self-forensics autonomic property (SFAP) is added to the ASSL toolkit to enable generation of the Java-based Object-Oriented Intensional Programming (JOOIP) language code with traces of Forensic Lucid to encode contextual forensic evidence and other expressions. The addition of self-forensics as an autonomic property is to augment the autonomic system specification language (ASSL) framework of formal specification tool for autonomic systems [12].

DMARF's components are stand-alone and may eavesdrop on RMI, XML-RPC, CORBA, and TCP connections for their requirements and do not follow SNMP protocol. As a result each achieved service will have to contain a proxy SNMP-aware agent regarding management jobs and provides instrumentation proxy to interconnect with the predefined services. DMARF is used as a base such that its services can be achieved through SNMP. MARF's proprietary management protocol that can be integrated with the use of common network service and device management, so the administrators can organize MARF nodes as a familiar protocol, as well as monitor their performance and gather the statistics or perform data manipulation [7].

In Figure 4, the relationship between all the major entities is defined in the system. Applications are the main manager while the remaining services could be both managers and agents sometimes. MARF services can communicate to each other and ask data from each other and they might display manager characteristics [7].

Java RMI, SOAP (Web-Services over XML-RPC), and CORBA has been implemented in DMARF, so some enhancement of works can be uprooted towards the management essentials. In particular, DMARF's CORBA IDL definitions can be utilized for SNMP agent generation. Thus, focus is on the role of CORBA in network management and the addition of DMARF's CORBA services implementation to deliver competent network management, monitoring, multimedia (audio) transfer, fault-tolerance and recovery, availability through replication, security, and configuration management [7].

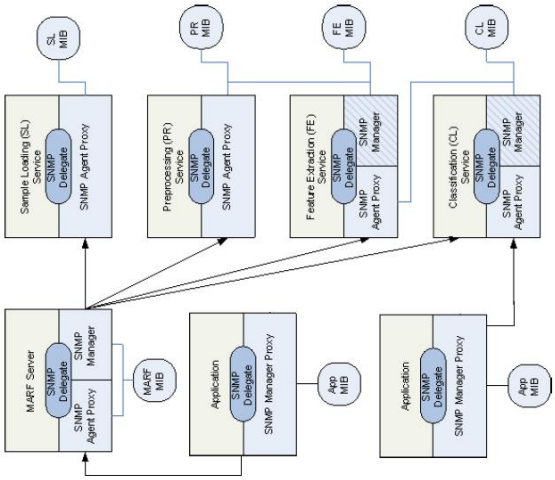

Figure 4. MARF Manager-Agent Architecture [7, p. 11]

2) GIPSY: GIPSY is an ongoing research platform developed at Concordia University. Its primary objective was to explore on a general resolution for the assessment of programs written in lucid intensional programming family of languages by means of a distributed demand-driven evaluation model. The system has been designed expending a framework methodology incorporating a lucid compiler framework and a demand-driven run-time system framework [10].

GIPSY is a distributed multi-tier and demand-driven framework. Essentially it contains a set of loosely coupled software components, which permits the evaluation of programs in a distributed demand-driven manner. The run-time system has three elementary objects: a tier, a node and an instance. GIPSY network is designed as an overlay network where network nodes and GIPSY tiers are systematized in a cluster called GIPSY instance. Node registration is done through a manager tier called the GIPSY Manager Tier (GMT) [10].

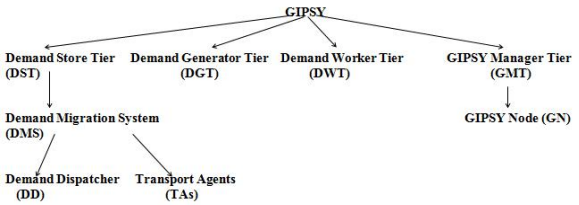

Figure 5. GIPSY Multi-Tier Architecture-Tree Structure

As shown in Figure 5, the multi-tier architecture of GIPSY has four distinct tiers: a DST, a DGT, a DWT and a GMT [13]. GIPSY run-time system inherits the following peer-to-peer network architecture principles: no single-point of failure, nodes and tiers can impeccably join or leave the network on the fly and demands are transmitted in such a way that place of process or storage is unknown to them [10].

GEE is a demand driven execution system that determines the control process by generating functional demands. GIPSY brings its users benefits of distributed programming. The architectural model of DMS consists of two parts, Demand Dispatcher (DD) and Transport Agents (TA). DMS relies on these two contributors to form a communication system similar to the e-mail delivery systems. The messages in the DMS are demands and results, each result being associated with one demand. DD and TAs implement the ability to migrate demands across heterogeneous system boundaries [14].

The following requirements must be fulfilled by a DMS: platform interoperability, once delivery semantics, asynchronous communication, no demands discrimination, no workers discrimination, secure communication, fault-tolerant demand migration, distributed technology independency, hot plugging and upgradability [14].

To plunge the complexity of the GIPSY through the technique of automation there is an architectural design approach named an autonomic version of GIPSY (AGIPSY). The main principle of AGIPSY is autonomic computing. AGIPSY is based on the specification model, which is built with ASSL [13].

AGIPSY architecture contains all the properties of a multi-agent loosely coupled distributed system with the virtues of decentralized control and allocation of data. So, its architecture enacts a non- hierarchical structure in which GNs (GIPSY Node) generate a medium for communication. The GN's plays an important role in the design of AGIPSY architecture because it is responsible for communication and it produces a distributed bridge for solving a problem. In AGIPSY, every GN needs the ability to plan and schedule tasks for the different tiers. The GNs are called GIPSY managers (GMs) [13].

The autonomous features of AGIPSY are: fault tolerance and recovery, self-maintenance, self-configuration, self-optimization, self-healing and self-protection. The complexity of GIPSY grows rapidly as it scales well widely. So to manage its complexity automatically is a tedious task for itself. Therefore an AC solution is required and it is generated in the form of AGIPSY, which makes GIPSY capable of self-managing [13].

GIPSY is taken as platform for compiling and evaluating Forensic Lucid. GIPSY uses Java-based Object Oriented Intensional Programming (JOOIP) which supports mixing Java and Lucid code. The ASSL toolset in the instance is to be clubbed with a code generation plug-in that generates JOOIP code with Forensic Lucid contextual expressions. GIPSY's GEE then evaluates the code [8].

After ASSL toolset generates the JOOIP code with Forensic Lucid fragments , it is passed on to the hybrid compiler of GIPSY, the GIPC to compile and link them together in a executable code inside the GEE engine resources which then would evaluate it by either of three models : the eduction model of GEE, Aspect J-based eduction Model and probabilistic model [8].

Forensic Lucid specifications are compiled to get states of the system under study in chronological order to understand the complex relations and events between components in the system. For event reconstruction and automatic deduction of computer crime incidents Forensic Lucid language is used as a forensic case specification language as it complements self-CHOP properties in autonomic computing [6].

As shown in Figure 6, GIPSY system consists of mainly 3 components: GIPC (General Intensional Programming Compiler) for compilation, GEE (General Eduction Engine) for execution and RIPE (Runtime Integrated Programming Environment). The communication between GIPC and GEE is done through GEER component. Here the distributed security totally relies on the underlying communication protocol i.e. RMI/RPC. In GIPSY system the evidences can be collected for forensics only if certified compilation is in place [6].

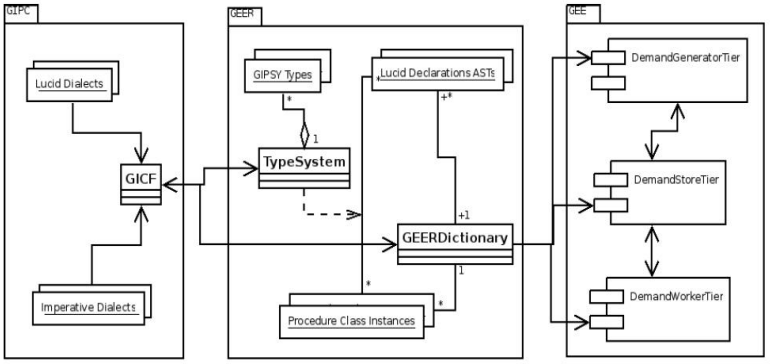

Figure 6. High level structure of GIPSY [6, p. 130]

For putting Self-Forensics into place and to collect evidence the GEE and GIPC components are the main ones in GIPSY, which will be requiring amendments. By further analysis, it was established that there is no way to do the forensic investigation of security incidents considering GIPSY system as a whole. Again, the evidence structure is quite similar as for other systems, which suggests that reusable models can be built for large number of similar systems for self-forensic

evidences [6].

GIPSY framework was designed in a segmental manner and has a lot of configurable components. So, it is very important that there's an automated solution for configuring and managing the deployment modules. Besides, previously a command-line interface was mainly used for managing the run-time system. A graph-based graphical user interface have been implemented which offers a set of user interfaces. Those interfaces enables users to unswervingly interact with the run-time system. The chief points are increasing the usability and empowering the user to have full control over the network with minimum manual interference. The tool have been tested in various platforms to confirm the portability [10].

GIPSY gives the integrated framework for running the programs in all version of lucid. This architectural framework follows the eductive model of computation. Intensional programs are examined using the lazy demand driven mode of execution called eduction in which program identifiers are computed in a point in space. Each demand is generated, executed and then stored as an identifier context pair [9].

First model for execution of lucid program was based on formal semantic of lucid and then, expanded for the implementation of the lucid interpreter, is known as eduction model. This eduction model can be defined as "tagged token demand driven dataflow" computing. Core concept of this model is notion of generation, propagation and consumption of demands and their computed values [9].

Lucid is independent of data types and algebra applied to the manipulated values. Programs can be executed in parallel but for better performance. More parallelism can be achieved by introduction of aggregated data types. GLU runtime system was created through generator worker architecture and implemented demand generation using a generator after eductive model of computation [9].

Architecture of GIPSY compiler,GIPC is framework based having components like parser, sematic analyser and translator. To run a hybrid program the architecture design should have the following elements: multi-tier architectural design, demand propagation and controller-based designs [9].

From the general requirements and design of the evaluation engine (GEE), Demand Migration Framework (DMF) was redesigned into two different fully integrated instances of its own framework, based on java distributed middleware technologies: Jini and JMS. In order to improve interoperability, remove flaws and allowing scalability testing, refactoring of parent DMF framework (DMS) of Jini and JMS is required [1]. Jini is a java-based technology for creating distributed systems, composed of Jini services and clients. It provides basic services such as lookup discovery and transaction services and, allows distributed Jini clients to read, write and remove serialized objects from shared object repository. Jini's DMS architecture is illustrated in Figure 7. JMS (Java Message Service) is Java Message Oriented Middleware technology constitutes API that allows application to create, read, send and receive messages. JMS DMS architecture is explained in Figure 8 [1].

To unify both DMF's, common functionality of two disjoint

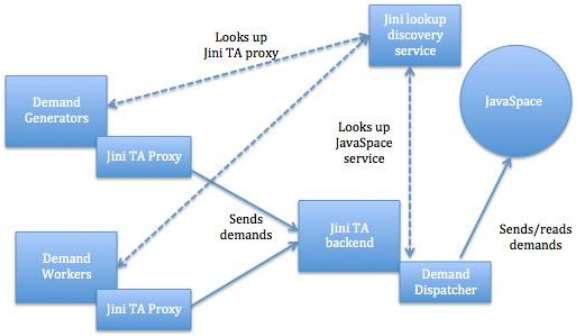

Figure 7. Jini DMS Architecture [8, p. 38]

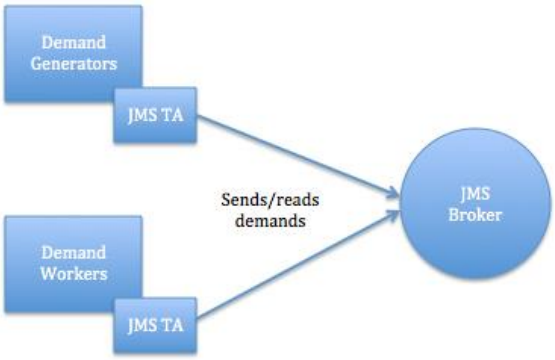

Figure 8. JMS DMS Architecture [8, p. 39]

DMF implementations were abstracted, wrappers were created to enclose them into common runnable module that can be initiated with single method call. JMS DMS and Jini DMS are created to be instances of same framework's implementation API, so that both DMS can be used interchangeably or concurrently with same GIPSY network instance [1].

III. SUMMARY

DMARF's RMI and CORBA implementation have been complemented with the Web Services for greater portability of MARF services over HTTP. The implementation work to build the Web Services feature was not high, but the placement within a servlet container and WSDL compilation needed substantial work. A graph-based GUI was implemented to simplify management of the GIPSY run-time system components. The solution depends on graph-based programming and visualization to denote a GIPSY network. It permits the user to seamlessly examine the status and properties of GIPSY nodes and GIPSY tiers at run-time other than aiming on increasing the usability.

ASSL framework is used for DMARF; to develop the autonomic self-management properties. Here we provided the methodology for the initial proof of concept. For the self-management in GIPSY, the AC is used to develop the AGIPSY which reduces the complexity of the system even while the system is scaled widely. So the autonomic properties are developed on the both open source systems and all these details

are verified by the different case studies and applications of DMARF and GIPSY.

Preliminary groundwork of requirements was placed to implement formally the self-forensics autonomic property within the ASSL toolset in order to allow any implementation of the self-forensics property. Studying the GIPSY system for the Self-Forensics we found out that the evidence structure is quite similar as for other systems. This suggests that reusable models can be built for large number of similar systems for self-forensic evidences.

Both DMARF and GIPSY systems should be allowed to add security mechanisms easily. But, it is not mandatory to have security layer to do large-scale scientific computation in controlled environment. JDSF as a proposed solution cover wide range of aspects such as data origin authentication, data integrity and malicious code detection. However, JDFS is not a complete solution towards security aspect. So, it is suggested to look forward for similar frameworks and enhance the security practices used in system.

We have analyzed codes for both GIPSY and DMARF on Windows 7 Home Premium 64- bit Operating system with processor Intel® Core(TM)2 Duo CPU T6600 @ 2.20GHz. We have analyzed the codes using Linux, CodePro Analytix, inCode and Metrics.

A. inCode Tool

inCode is the lightweight and affordable quality assessment tools for Java, C++ and C. It is used for evaluating the quality on the level of code and design. It finds bad smells, such as 'Code Duplication', 'Data Class' and 'Data Clumps', in the code.

inCode saves most of the overhead during lengthy code review and restructuring sessions. Also, inCode reduces the time needed to understand the context of any given problem, since the problem affects the code you are already working on [15].

Installation:

We installed the inCode tool from the following link: http://www.intooitus.com/products/incode/download

Usage:

Steps to analyse code using inCode Tool are following:

1) We clicked on Analyze Java Project.
2) Then select the Source location and click run.
3) Output will give counted number of classes and methods.

B. CodePro Analytix Tool

CodePro Analytix tool is a software code quality, testing and static analysis tool to improve software quality, security, reliabilty and maintainability of java applications[16].

Installation:

1) We go to Help and then Click on Install New Software to install the plug-in for this tool.
2) Put the proper link under "Work with" and Click on 'Add' button. The link is http://dl.google.com/eclipse/inst/codepro/latest/3.7
3) Click Next button and eclipse will install CodePro Analytix.
4) Accept license while installing plug-in.
5) In case of warning message, press **OK** to continue and restart eclipse after installation.

Usage:

Steps to analyse code using codePro Analytix are following:

1) Right click on Source code of project and select Compute Metrics (**CodeProTools** ->**Compute Metrics**).
2) In Console we have seen Number of Lines.
3) We have Founded Number of Class and Number of Methods by clicking Number of Types and Number of Methods Respectively and then we clicked on Breakdown by scope.

C. Metrices

Installation:

1) We go to Help and then Click on Install New Software to install the plug-in for this tool.
2) Put the proper link under "Work with" and Click on 'Add' button. The link is http://metrics.sourceforge.net/update [17].
3) Click Next button and eclipse will install Metrices.
4) Accept license while installing plug-in.
5) In case of warning message, press **OK** to continue and restart eclipse after installation.

Usage:

Steps to analyse code using Metrices are following:

1) Click Project ->Properties ->Enable Metrics.
2) we can view metrics result using **Metrics View** (**Windows** ->**Show View** ->**Other** ->**Metrics View**)
3) So every time when program is compiled we could see the Output in Console.

The results of above described tools are given below:

Table I
CODE ANALYSIS FOR DMARF

Measures	Linux	CodePro Analytix	incode	Metrics
Number of Java Classes	1058	983	1058	979
Number of Java Files	1024	-	-	-
Number of Java Methods	-	6305	7554	6109
Number of lines of Java	131706	77297	-	77297

DMARF's RMI and CORBA implementation have been complemented with the Web Services for greater portability of MARF services over HTTP. The implementation work to build the Web Services feature was not high, but the placement within a servlet container and WSDL compilation needed

Table II
CODE ANALYSIS FOR GIPSY

Measures	Linux	CodePro Analytix	incode	Metrics
Number of Java Classes	705	626	702	580
Number of Java Files	602	-	-	-
Number of Java Methods	-	5680	6468	5746
Number of lines of Java	139632	104073	-	104073

substantial work. A graph-based GUI was implemented to simplify management of the GIPSY run-time system components. The solution depends on graph-based programming and visualization to denote a GIPSY network. It permits the user to seamlessly examine the status and properties of GIPSY nodes and GIPSY tiers at run-time other than aiming on increasing the usability.

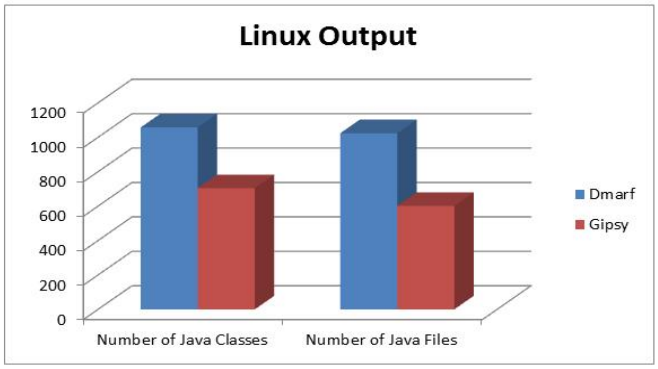

Figure 9. Linux Output for Java Classes and Files

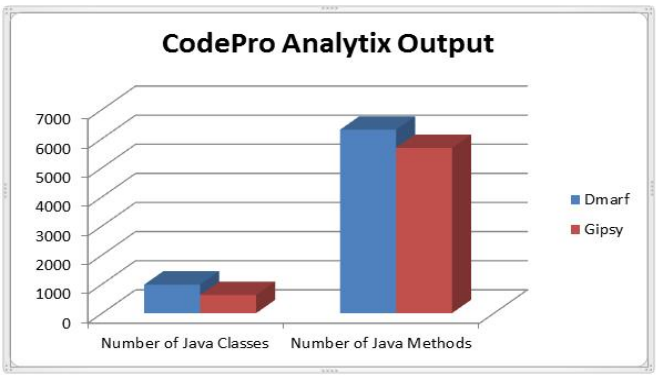

Figure 10. CodePro Analytix Output for Java Classes and Methods

ASSL framework is used for DMARF; to develop the autonomic self-management properties. Here we provided the methodology for the initial proof of concept. For the self-management in GIPSY, the AC is used to develop the AGIPSY which reduces the complexity of the system even while the system is scaled widely. So the autonomic properties are developed on the both open source systems and all these details

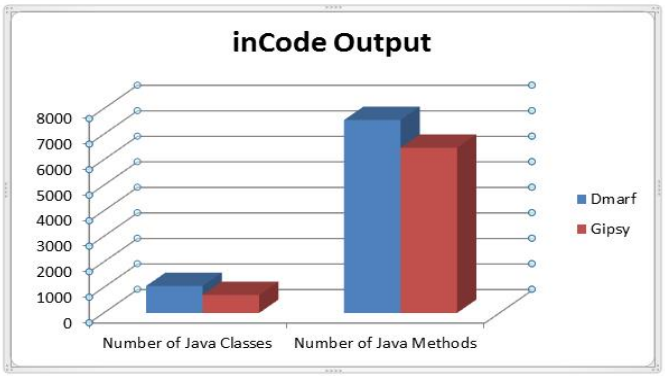

Figure 11. inCode Output for Java Classes and Methods

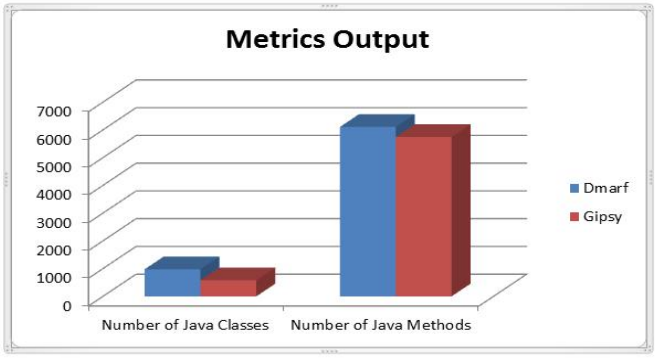

Figure 12. Metric Output for Java Classes and Methods

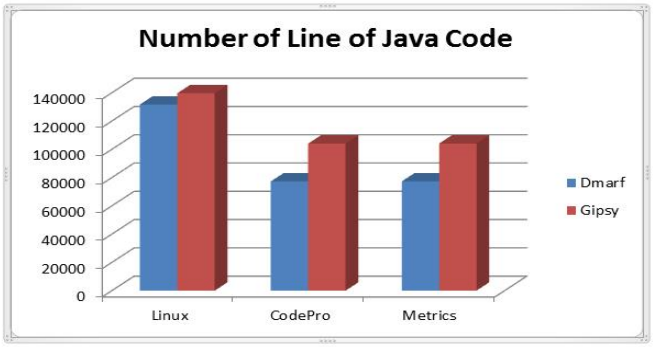

Figure 13. Number of Lines of Java Code

are verified by the different case studies and applications of DMARF and GIPSY.

Preliminary groundwork of requirements was placed to implement formally the self-forensics autonomic property within the ASSL toolset in order to allow any implementation of the self-forensics property. Studying the GIPSY system for the Self-Forensics we found out that the evidence structure is quite similar as for other systems. This suggests that reusable models can be built for large number of similar systems for self-forensic evidences.

Both DMARF and GIPSY systems should be allowed to add security mechanisms easily. But, it is not mandatory to

have security layer to do large-scale scientific computation in controlled environment. JDSF as a proposed solution cover wide range of aspects such as data origin authentication, data integrity and malicious code detection. However, JDFS is not a complete solution towards security aspect. So, it is suggested to look forward for similar frameworks and enhance the security practices used in system.

IV. Requirements and Design Specifications

A. Personas,Actors and Stakeholders

1) DMARF: The description of persona is as follows:

Primary Persona	Developer
Faculty	Bhavin Patel
Department	Computer Science and Software Engineering
Organization	Roorkee University
Education	PhD.

Bhavin was born in the city of Ahmedabad in India. He had lived in Nadiad and, mostly in the town of Kheda since the age of 5. He finished his primary, secondary, and high schools from there as well. He did his Bachelors at university of Niper. Bhavin had completed two Masters, one in Computer Science and another in Information Systems Security at the same university in 2005 and in 2007. He has completed his PhD in 2013 from Roorkee.

Bhavin has worked in the development team to make DMARF (Distributed MARF). The distributed MARF enhances the following to MARF: coping with the extremely higher demand of users in processing power and data storage, high availability, scalability and resistant to failure.

Relevant Skills:

- Knowledge about the platform.
- Understanding of the domain.
- Know the development process.
- Know how to do effective estimations.
- Proficiency on the programming languages.
- Know how to maintain a project over time.

Goals:

- Making resources accessible.
- Constructing a collection of computers appears as a single computer.
- Hiding all the distribution from the users as well as the application programs.
- Making the system easier to build and change..
- Distributed system should be more reliable than single system.
- The system should be scalable and performance should be upright.

2) DMARF: The description of persona is as follows:

Primary Persona	Researcher
Faculty	Abhijit Kuch
Department	Computer Science and Software Engineering
Organization	Jordar University
Education	PhD.

Abhijit has done research on various aspects of MARF and then how to make MARF distributed. MARF is an open source research platform where pattern recognition, natural language processing and signal processing algorithms are assembled and written in Java. He has researched on to add much greater interoperability of the DMARF nodes over the Internet and restricted environments where HTTP is the only protocol allowed. For this, Web services need to be integrated to DMARF.
Research Interests: Abhijit's research interests involve different aspects of the GIPSY system as well as MARF and various areas such as distributed and parallel computing, languages and models; AI, pattern recognition, NLP, and machine intelligence, computer graphics, and information-systems security.

Relevant Skills:

- Subject knowledge.
- Research methods – theoretical knowledge and practical application.
- Information seeking, literacy and management.
- Project planning and delivery.
- Analysing, synthesising and critical thinking.
- Commitment to research and time management.
- Responsiveness to change and work-life balance.

Goals:

- Research for designing and implementing web services for DMARF.
- Help other researchers and developers to make sense of information.
- To build, experiment and develop algorithms and systems according to requirements.

3) DMARF: The description of persona is as follows:

Primary Persona	Application Developer
Application Developer	Aditya Dewal
Department	Computer Science and Software Engineering
Organization	Concordia University
Education	B.Tech

Aditya is involved in a project at Concordia University who got a contract from Ontario Police Department to develop a Text-Independent Speaker

Identification Application. Text-independent systems are mostly used for speaker identification, as they need very little if any support by the speaker. Here, the text during enrolment and test is different. The enrolment can happen without user's awareness, which occurs for many forensic applications. Aditya is going to use the DMARF framework to develop the application, which will do the task of determining an unknown speaker's identity.

Relevant Skills:

- Analytical and problem solving.
- Good in adapting new features to existing application.
- Conduct testing and installation of application.
- Generating Documentation and User Manuals.

Goals:

- Deliver application as per client expectation.
- Need to be highly maintainable in order to maximize efficiency, reliability, and safety.
- Deliver high usability, as the application needs to be intelligible and interactive.

4) DMARF: The description of persona is as follows:

Primary Persona	Police Investigator
Investigations Bureau Commander	Kanwaldeep Singh
Department	Investigations Bureau
Organization	Ontario Police Department
Education	BA in Criminology

Captain Kanwaldeep began his law enforcement career in 1999 with the Ontario Police Department. He was born on December 13, 1977. He is a fifteen-year veteran of law enforcement and holds a Bachelor's Degree in Criminal Justice from Ktarnak University. After graduating from the San Bernardino County Sheriff's Academy, Captain Daya worked as a Patrol Officer. He was later selected to start the Bicycle Program at the Ontario Police Department. During his career He has been involved in every type of criminal investigation at the local, provincial and federal levels. He now brings the techniques learned during his time on those task forces to investigations of crimes committed using the Internet.

Captain Benson is looking for a web services system that can be used on a desktop for high volume processing of recorded audio, imagery or textual data for biometric forensic analysis and pattern recognition. He wants to process recorded phone conversations for subject identification and classification.

Relevant Skills:

- Ability to identify and analyze social problems and develop solutions.
- Broad understanding of criminal law and the criminal justice system.
- Computer literacy.
- Critical thinking and decision-making.
- Interviewing skills and knowledge of legal structures.
- Quantitative skills and research strategies.
- Skills in research and scientific methodology.
- Understanding nature of crimes and societies' reaction to crimes.

Goals:

- Processing high volume of recorded data for biometric forensic analysis and pattern recognition.
- Making sure that data gets documented and preserved.
- Getting the results for subject identification and classification.
- Protecting the privacy of the recorded audio, imagery or textual data.

5) DMARF: The description of stakeholder is as follows: Ashish is given the task of researching on DMARF case

Stakeholder	Student
Graduate Student	Ashish Arora
Department	Computer Science and Software Engineering
Organization	Concordia University
Education	B.Tech

studies and come up with an enhancement, which would help the framework to be more usable. He goes through the case studies and identifies some of the limitations of the system. Some features like WAL (Write Ahead Log – write ahead message-logging) is partially developed. He plans on working with the WAL protocol.

The protocol is designed for DMARF for the disaster recovery of uncommitted transactions and for avoiding data loss. His goal is to contribute towards DMARF research activities through collaboration, motivation and cooperation. By developing the full features of the protocol, Ashish is hoping to make the framework more effective and efficient.

6) DMARF: The description of stakeholder is as follows: Dr. Samy has interests on different research areas. He

Stakeholder	Professor
Faculty	Samy Carton
Department	Computer Science and Software Engineering
Organization	LDRP University
Education	Ph.D

has been working with MARF since 2002. He wants his students to study DMARF (Distributed MARF) articles and if possible make some enhancement to the framework. His

main prospective is to provide the basis of the system to the students so that they can explore the system and make it more prolific. He has asked the students to profoundly analyse the case study of DMARF with some quality analysis tools, in order to make the framework more structured. Students can either implement the missing features or refactor the code of the current version of the framework. He has also asked them to investigate the architectural design of the system, through which overall design of framework can be ameliorated.

7) GIPSY: The description of persona is as follows:

Primary Persona	Developer
Research Fellow	Mukesh Kumar
Department	British Software Engineering Research Centre
Organization	Concordia University
Education	Ph.D.

Mukesh received his M.Sc. in Computer Science (2005) and Ph.D. in Computer Science (2008) from Concordia University, Canada. Currently, he is a Research Fellow at Pero (the British Software Engineering Research Centre) at University of Mcgill. His current research focuses on knowledge representation and self-awareness for self-adaptive systems. His research interests are in software development methodologies for developing autonomic systems.

Mukesh has worked on the implementation of GIPSY's Demand Migration System (DMS). This system makes GIPSY highly distributive and interoperable by mixing together advanced distributed technologies. As GIPSY nodes are located at different machines DMS should integrate a secure mechanism. There must be concern about behavior of the system if some fault occurs and it should be generic enough to allow the use of other technologies which are not part of its original implementation.

Relevant Skills:

- Complex problem solving.
- Knowledge about the domain.
- Systems analysis and evaluation.
- Operations analysis and critical thinking.
- Programming.

Goals:

- Platform interoperability and once delivery semantics.
- Asynchronous Communication.
- No demands discrimination and no workers discrimination.
- Distributed Technology independency.

Primary Persona	Researcher
Associate Professor	Maja ave
Department	Computer Science and Software Engineering
Organization	Concordia University
Education	Ph.D.

8) GIPSY: The description of persona is as follows:

Maja completed his college diploma in Pure and Applied Science in 1989, after that he did his bachelors in Computer Science at University of Concordia. He pursued his Master of Computer Science in 1995 from the same university. Furthermore, he finished his PhD in Computer Science in Intensional Scientific Programming at University of Montreal in 1999. His personal interests are history of humanity, history of science, history of technology, military history, Philosophy and sociology. He likes reading, listening to music, hiking, camping and playing games.

Maja has done research on various aspects of intensional programming. He is one of the researchers working on the GIPSY platform. Maja is researching on the development of a graph-based graphical user interface for GIPSY which will upsurge the usability and allow the user to have full govern over the network with least manual intervention.

Research Interests:

Maja's professional interests are design and implementation of programming languages, parallel/distributed programming languages and execution architectures, intensional programming, software engineering and software process.

Relevant Skills:

- Subject understanding and research methods.
- Critical thinking and problem solving.
- Inquiring mind and intellectual insight.
- Innovation and argument construction.
- Responsiveness to change and work-life balance.

Goals:

- Research for designing and implementing a graph-based graphical user interface for GIPSY.
- Offer a set of user interfaces through graph-based GUI.
- Enable users to unswervingly interact with the run-time system.

9) GIPSY: The description of persona is as follows:

Primary Persona	Application Developer
Application Developer	Jack Mak
Department	Computer Science and Software Engineering
Organization	Cornell University
Education	PhD.

Mukesh is working on a project at Concordia University where they are building an application for self-forensic enabled unit. The unit will help inquiry of incidents and also automated reasoning and verification of theories beside the events reconstruction.

This will also help keeping forensics data for additional analysis of reports crashes, failures and anomalies. Mukesh plans to use the GIPSY framework as the evaluation system which can verify the theory automatically against the context of evidential statement. This system will improve the safety of the passengers and their vehicles.

Relevant Skills:

- Design solution for anticipated problem.
- Critical thinking for examining application.
- Refactoring existing application.
- Optimizing system performance.
- Perform user acceptance testing.

Goals:

- Deliver the functions essential for the customer.
- Provide efficiency and security.
- Respond within acceptable limits, as result dissatisfaction will be minimal.
- Should be maintainable and reliable.

10) GIPSY: The description of persona is as follows:

Primary Persona	Application Developer
Aircraft Accident Investigator	Daya Darwaja
Department	Air Investigation
Organization	Transportation Safety Board of Canada
Education	B.Tech

Daya have been working in Transportation Safety Board of Canada for 15 years. He graduated from Concordia University in Mechanical Engineering. He is a certified pilot. He has diverse and widespread experience in the aviation industry. As well as conducting investigations, he have participated in national and international government and industry groups to monitor safety trends and communicate safety issues to change agents.

Relevant Skills:

- Signifying knowledge of civilian aircraft design, manufacture and maintenance operations or civilian aircraft operational requirements, practices and procedures.
- Experience as a pilot for scheduled air carrier and certified flight instructor.
- Strong written and communication skills, inquisitive, ability to empathize and know your audience.

11) GIPSY: The description of stakeholder is as follows:

Stakeholder	Student
Graduate Student	Lovepreet Singh
Department	Computer Science and Software Engineering
Organization	Concordia University
Education	M. Eng.

Lovepreet comprehends the functionality of GIPSY framework, from various research theories and practices, to explore various applications of GIPSY. He studied about its architecture, components and interaction between components to get the mental model of GIPSY. He also works on finding the limitations of GIPSY framework and what steps could be taken to overcome these them. He tests the framework's security features for any loopholes, along with how security layer having configuration parameter can be implemented on GIPSY and explores the overheads of implementing the security layer.

12) GIPSY: The description of stakeholder is as follows:

Stakeholder	Professor
Graduate Student	Saras Cha
Department	Computer Science and Software Engineering
Organization	Concordia University
Education	Ph.D

Dr. Saras has research interests in context-driven computing, intensional programming, demand-driven computing, design and implementation of programing languages. He pursues students to have knowledge about GIPSY architecture by reviewing the case study of GIPSY using various analysis tools. Students will go through the architectural design of the system, understand different actors of the system to make it more versatile.

B. Use Cases

1) DMARF: The use case are very essential for the domain modeling. Students will go through the architectural design of the system, understand different actors of the system to make it more versatile.

Supposition: Police Investigator is the experienced user of DMARF application and gives correct inputs and data such that no anomaly or abnormal behaviour is encountered during the execution of the use case.

Table III
USE-CASE FOR DMARF

USE Case UC1	**Process File**
Brief Description	This use case describes how DMARF can be used for processing files.
Scope	System
Level	User Goal
Primary Actor	Police Investigator
Stakeholders and Interests	• **Police Investigator** Wants accurate processing of uploaded file.
Preconditions	1) Police Investigator is identified and authenticated. 2) General MARF services are appending running as Distributed MARF web services.
Minimal Guarantee	WAL (Write-Ahead-Log) logs the transactions.
Success Guarantee (Postconditions)	Uploaded file is **processed** successfully and correct **result-set** is provided to the user.

Main Success Scenario (or Basic Flow)	1) **User** uploads **audio or image file** for pattern recognition (processing). 2) System does **preprocessing** on the uploaded file by applying processes like filter, normalization etc. 3) **Preprocessed** data is extracted by **Feature Extraction Service** and attempts to extract certain features as requested by user. 4) **Feature extraction services** might also query the existing preprocessed data in the database. 5) Feature details from **feature extraction service** are accepted by **Classification and Training service**. 6) The same service either updates **training database sets** or performs **classification** against existing database sets. 7) **Feature Extraction Service** might be queried optionally by this service as well.
Extensions (or Alternative Flows)	2a. At any time, System fails: 1) The system failure initiates recovery where it communicates with the replicas and by scheme of gossiping and **logs** analysis, does recovery of the state to correct state. 2) To support recovery from automated transaction **log**, logs are recorder at pre-stored checkpoints. 3) The transaction logs contain details about the transactions and the timestamp of their execution. This transaction logging technique is called WAL.
Special Requirements	• System should allow bulk processing and provide correct results. • Quality attributes of the system include usability, reliability, performance, and supportability requirements.
Technology and Data Variations List	None
Frequency of Occurrence	Could be nearly continuous.
Open Issues	• Transaction IDs recycle conditions are not specified. • Large numbers of open log files causing system to crash. • Design flaw due to few nested functionalities in modules. • Concurrency yet to be refined. • There is no limited size of garbage collection for WAL. • Less reliability and availability due to no primary backup or replication.

2) GIPSY: The use cases plays an important role here again.

Supposition: Aircraft Accident Investigator is the proficient actor of GIPSY application and provides accurate inputs and data so that no incongruity or anomalous conduct is faced during the execution of the use case.

Table IV
USE-CASE FOR DMARF

USE Case UC2	**Evaluate Evidence**
Brief Description	This use case describes how Self-Forensic can be used in Flight-Critical system using GIPSY.
Scope	System
Level	User Goal
Primary Actor	Aircraft Accident Investigator
Stakeholders and Interests	• **Aircraft Accident Investigator** Engineer defines theories as observation sequence about the happened incident. Evaluate sequence against collected forensic evidence.
Preconditions	1) Aircraft Accident Investigator is identified and authenticated. 2) Self-forensic sensors should be operational.
Success Guarantee (Postconditions)	1) Self-forensic sensors record and process consistent and accurate events. 2) Evidences would be evaluated against existing data to provide correct forensic evaluation result.

Main Success Scenario (or Basic Flow)	1) **Aircraft accident investigator** provides **hypothesis** to **GEER**. 2) **GEER** observes for the **evidential statements**. 3) **GEER registers GIPC** and allocates all the required theoretical observations. 4) **GIPC** is responsible to negotiate with **preprocessor**, **forensic lucid compiler** and **general eduction engine** components. 5) **Preprocessor** plays the role for the **preprocessing** of intensional programming (lucid program). 6) Lucid program is the key input for **forensic lucid compiler** to perform multi-dimensional context analysis. 7) The output of the compiler will store in **GEE**. 8) **GEE** is responsible to generate the **processed result**. 9) And the result is extracted by the **aircraft accident investigator**.
Extensions (or Alternative Flows)	3a. At any time, self-forensic components fail 1) Duplicate self-forensic components (sensors) will be used.
Special Requirements	• Alternate input from sensors, human, logs. • Preserved forensic data must be atomic, robust, reliable and durable. • Should have enough tools for automated reasoning and reporting about incident analysis matching.
Technology and Data Variations List	None
Frequency of Occurrence:	Could be nearly continuous.

Open Issues	• The cost of overall system will increase. • Design and development require long-term and consistent storage and system power. • Design flaw due to few nested functionalities in modules. • Self-forensic logging and analysis should be done in ROM or similar type of memory and should be upgradable.

C. Domain Model UML Diagram

1) DMARF: In DMARF system the MARF pipeline components are wrapped into distributed classes (services) to introduce the distributed functionality into the system. In DMARF each stage will have a replica manager, working replicas which will coordinate with the server frontend to work with the samples inputted/loaded by the user and provides distributed features like data recoverability, availability and efficiency.

When a request comes from the police investigator by providing the file, it is then delivered to sample loader which in turn will decide what is to be done with the file. Sample loader might provide the file for pre-processing or may provide it for feature extraction if it's already preprocessed. After feature extraction the classification of the features is done and then the extracted featured can be classified and stored into the training database or might be queried for finding existing features in the database to provide results to the logged in user. These components are wrapped into interface to provide remote feature to the application. At the end the processed result which can be either a training set or a result set would be brought to the police investigator.

2) GIPSY: In the domain model the bottom up and top down approaches have been combined. Here the multi-tier architecture consists of layers as GIPC, GEE and GEER. GIPC has a one to one relationship with GEER where the GIPC registers to GEER for the compilation of evidential statements with the story.

GIPC in-turn has a one to one relationship of preprocessing with preprocessor to preprocess the evidences into forensic lucid format, one to one compiling relationship with ForensicLucidCompiler to compile the Forensic lucid statements and one to one converts relationship with GEE to store the results.

GEER is associated with evidential statements to get the stories related to the accident. Further the GEE is associated to ProjectedResult which in turn is associated with AircraftAccidentInvestigator to provide result.

3) Fused DMARF-over-GIPSY Run-time Architecture (DoGRTA): The concept here is to merge the concept of two architectures in one architecture and fused the pipeline concept of DMARF over the muti-tier concept of GIPSY. So the fused

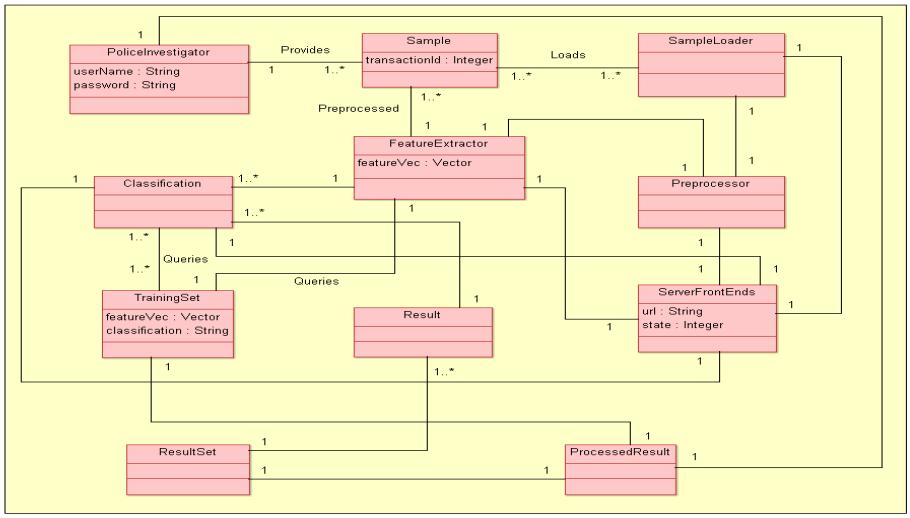

Figure 14. DMARF Domain Model

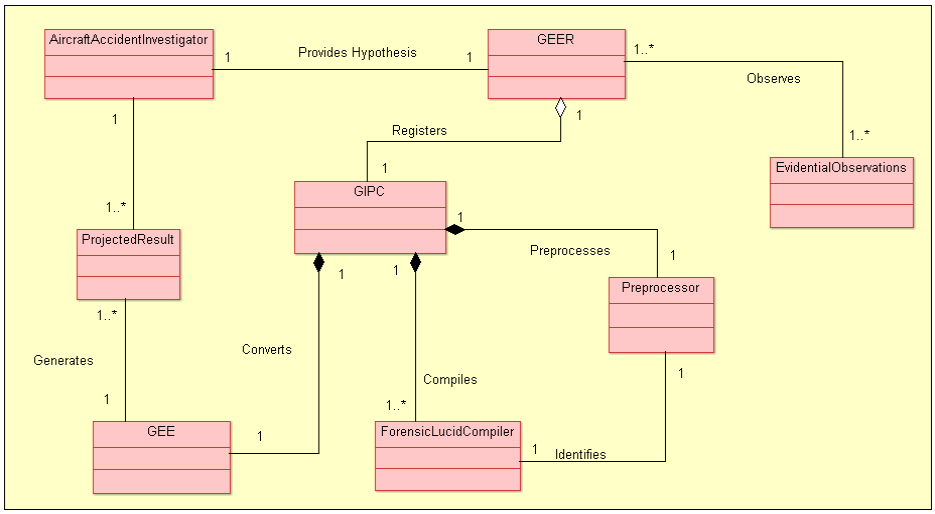

Figure 15. GIPSY Domain Model

architecture will provide overcome overfailures of DMARF architecture.

For overlaying GIPSY feature over DMARF feature we divided the stages of DMARF to overlap the stages in GIPSY i.e. the stages of pipelines will behave like pipelines inside a tier but the pipelines have been broken up in different tiers to incorporate the multi-tier architecture of GIPSY as well. For attaining this architecture GIPC is the entry point of data in the pipeline which then is associated with the PreprocessorGenerator to do the preprocessing necessary on the input data. For the input data the request is generated using ProceduralDemand, IntentionalDemand, SystemDemand or ResourceDemand and then is respectively allocated to the worker tier(ProceduralDWT, IntentionalDWT, SystemDWT or ResourceDWT) on the basis of request type. The working Tier further delegates the request to the SampleLoadingWorker which loads the sample to the pipe line of DMARF for the further processing of FeatureExtractionDemands, ClassificationDemands and further the result is extracted as TrainingSet or ResultSet and provided to DST.

So, ephemeralness of DMARF over GIPSY is derived in the form of intra-process communication between run-time multi-tiers.

D. Actual Architecture UML Diagrams

1) DMARF: In the conceptual domain diagram we have considered SampleLoader to be a single point of entry to the pipeline and rest stages comes later whereas in the actual system MARF factory class is present which delegates control to the correct module on the basis of the inputs given. We have also considered that the result could be obtained from either Classification or from FeatureExtraction class whereas in the actual code the ResultSet class only has the association with Classification module.

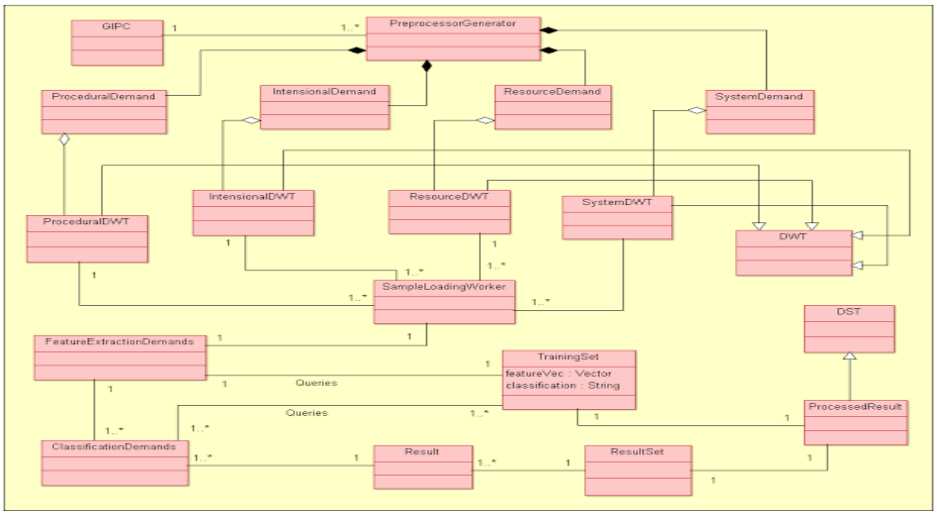

Figure 16. DMARF-over-GIPSY Fused Domain Model

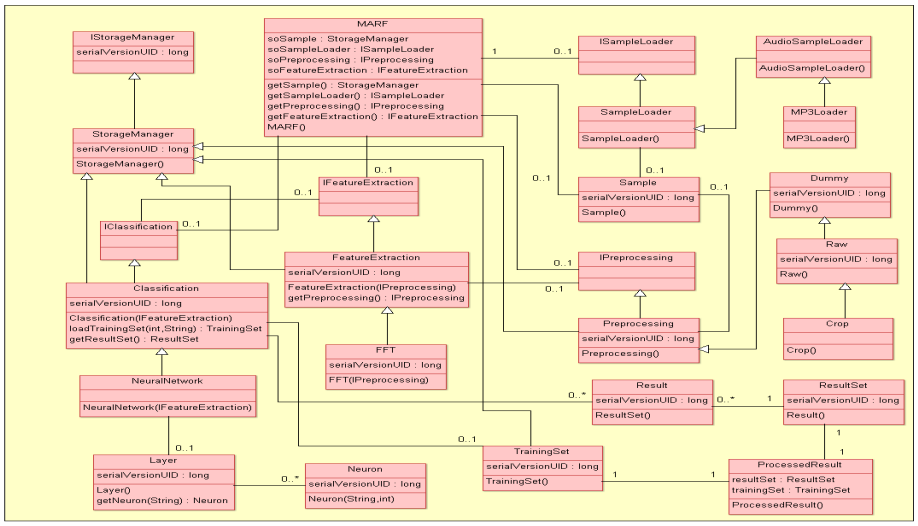

Figure 17. Actual Architecture UML Diagram of DMARF

We supposed that the FeatureExtraction class would query the Training database and can also give the results before classification is done which is incorrect according to actual class diagram. Further in the conceptual diagram the association between the FeatureExtraction and the TrainingSet is shown. This we did to represent the fact that the FeatureExtraction module can also query the Training database for details. This association is not depicted in the actual class diagram.

Further, all the associations that we have done between the conceptual classes are 1..*, considering that the next stage will only work when the input arrives with the subsequent stages which is not actually true as according to the actual class diagram the stages of DMARF pipeline as having association of 0..* which signifies that the stages can work independent of the fact that 0 or no input was provided by the subsequent stage. In the conceptual model the Preprocessor class is mapped to Preprocessing class in the Actual class diagram. The FeatureExtraction class is mapped to FeatureExtractor class in the conceptual domain diagram. For rest of the classes the names are the same. Police Investigator class in the Conceptual class diagram represents the class to handle the inputs by the user which is not present in the Actual diagram and instead the MARF class acts as a factory class to handle inputs by the user and direct to the correct stage of the pipeline.

2) GIPSY: One of the major differences in the conceptual and actual Domain diagram is that the PreprocessorParser and Preprocessor are inherited by Preprocessor-ParserConstants which is not drawn in the domain diagram because we considered that parsing is done by Preprocessor only as an independent entity.

Preprocessor has association with SimpleNode as one to one relationship but in domain diagram we considered it as 1 to 1* relations because we assumed that a single Preprocessor is involved with multiple SimpleNode for the registration and

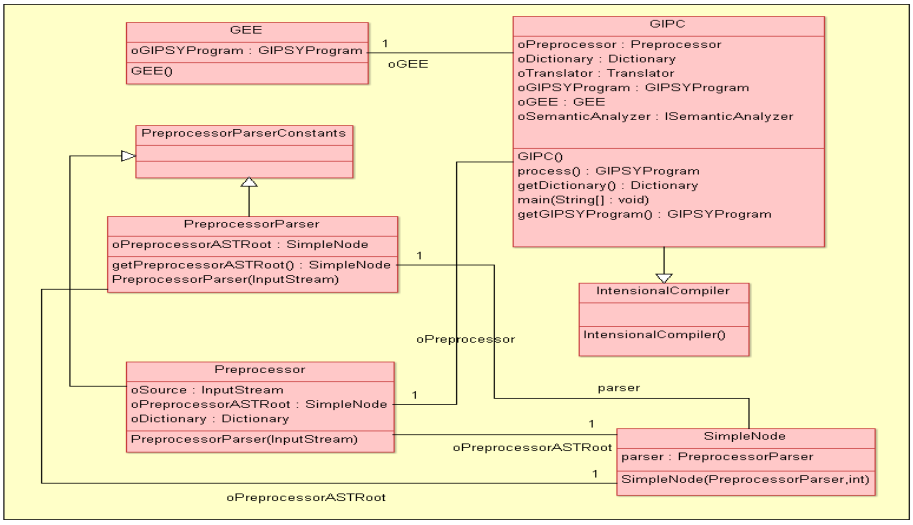

Figure 18. Actual Architecture UML Diagram of GIPSY

allocation. Further IntentionalCompiler mapped as ForensicLucidCompiler does not have association with Preprocessor in the actual class diagram.

3) Two Classes and the Relationship between them: Layer Class:

```
Layer Class:

package marf.Classification.NeuralNetwork;
public class Layer extends BaseThread implements Serializable{

public Layer(){
            super();
      }
}
```

Figure 19. Layer Class

NeuralNetwork Class:

```
NeuralNetwork Class:

package marf.Classification.NeuralNetwork;
public class NeuralNetwork extends Classification{
private ArrayList<Layer> oLayers = new ArrayList<Layer>();
private Layer oInputs = new Layer();
     private Layer oOutputs = new Layer();

public final void generate(int piNumOfInputs, int[] paiHiddenLayers, int
piNumOfOutputs) throws ClassificationException
     {
          Layer oHiddenLayer = new Layer();
          for(int iCurrentLayer = 0; iCurrentLayer < this.oLayers.size() - 1;
iCurrentLayer++){
               Layer oTmpLayer = (Layer)this.oLayers.get(iCurrentLayer);
               for(int n = 0; n < oTmpLayer.size(); n++){
                    Neuron oCurrentNeuron = oTmpLayer.get(n);
                    Layer oNextLayer = (Layer)this.oLayers.get(iCurrentLayer
+ 1);
               }
          }
     }
```

Figure 20. NeuralNetwork Class-Part1

```
public final void train(final double[] padInput, int piExpectedLength, final
double pdTrainConst) throws ClassificationException{
     for(int i = this.oLayers.size() - 2; i >= 0; i--){
          Layer oLayer = (Layer)this.oLayers.get(i);
          oLayer.train(pdTrainConst);
          }
     }
     public final void commit(){
          for(int i = 0; i < this.oLayers.size(); i++)
          {
               Layer oLayer = (Layer)this.oLayers.get(i);
               oLayer.commit();
          }
     }
}
```

Figure 21. NeuralNetwork Class-Part2

4) ObjectAid UML Explorer Tool: We used ObjectAid tool to Class diagrams of DMARF and GIPSY [28]. Steps to install and use ObjectAid, and for creating class diagram that we used are illustrated below:

Installation:

1) We go to Help and then Click on Install New Software to install the plug-in for this tool.
2) Put the proper link under "Work with" and Click on 'Add' button. The link is http://www.objectaid.net/update
3) Accept license while installing plug-in.
4) In case of warning message, press **OK** to continue and restart eclipse after installation.

Usage:

1) Create an empty class diagram with the '**New**' wizard.
2) Drag selected class onto your diagram from **Project Explorer**.

3) Right click on selected class to add or remove **Dependencies** in class diagram.

5) ArgoUML Tool: We used ArgoUML tool to create Class diagrams of DMARF and GIPSY [21]. Steps to install and use ArgoUML, and for creating class diagram that we used are illustrated below:

Installation:

1) We have downloaded ArgoUML from the following link: http://argouml.en.softonic.com/

Usage:

1) In ArgoUML, create new diagram (**File** ->**New** ->**Create** ->**New Class Diagram**)
2) Mostly we have used the following options: **New Class**, **New Association**, **New Composition**, **New Generalization** and **Rectangle**.
3) To get output image, we used Export Graphics option (**File** ->**Export Graphics**).

V. Methodology

A. Refactoring

1) Identification of Code Smells and System Level Refactorings: We have identified some of the code smells by using tools like JDeodorant and CodePro Analytics. To find the rest we manually went through the source codes. To find out how the bad smells and their solutions affect the design we have used ObjectAid.

2) DMARF: **Code Smell: Issue A**

Package	marf.Storage.Loaders
Source Class	TextLoader
Method	readSampleData(double[]): int
Code Smell	Switch Statement
Code Lines	121 - 240

Motivation

One of the most obvious symptoms of object-oriented code is its comparative lack of switch (or case) statements. The type code affects the behavior of a class. The object-oriented notion of polymorphism gives us an elegant way to deal with this problem. We will replace the type code with a state object. We will use Replace Type Code with State/Strategy for the above case.

Mechanics

1) We self-encapsulate the type code.
2) We create a new class, and name it after the purpose of the type code. This class is the state object.
3) Then we add subclasses of the state object, one for each type code.
4) After that we create an abstract query in the state object to return the type code. Subsequently, create overriding queries of each state object subclass to return the correct type code.
5) Next is compiling.
6) Next is to create a field in the old class for the new state object.
7) We adjust the type code query on the original class to delegate to the state object.
8) And then adjust the type code setting methods on the original class to assign an instance of the appropriate state object subclass.
9) At the end compile and test.

The solution will affect the system design as we are introducing four new classes to the package.

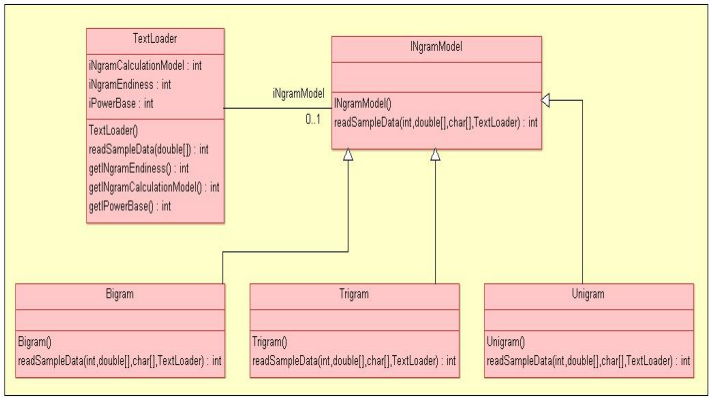

Figure 22. Partial Class Diagram for Issue A

DMARF Code Smell: Issue B

Package	marf.Classification.NeuralNetwork
Source Class	NeuralNetwork
Method	interpretAsBinary()
Code Smell	Feature Envy
Target Class	Layer

The left class in the diagram is NeuralNetwork and the right one is Layer. The method interpretAsBinary is in NeuralNetwork but it is more coupled with Layer.

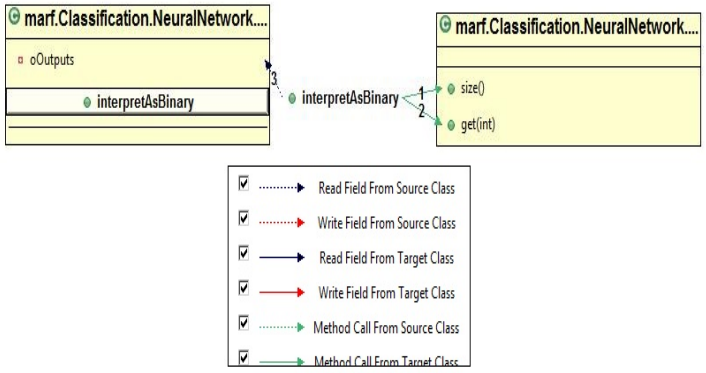

Figure 23. Code Smell Visualization

Code Smell: Issue C

Package	marf.Classification.NeuralNetwork
Source Class	NeuralNetwork
Method	getOutputResults()
Code Smell	Feature Envy
Target Class	Layer

The left class in the diagram is NeuralNetwork and the right one is Layer. The method getOutputResults is in NeuralNetwork but it is more coupled with Layer.

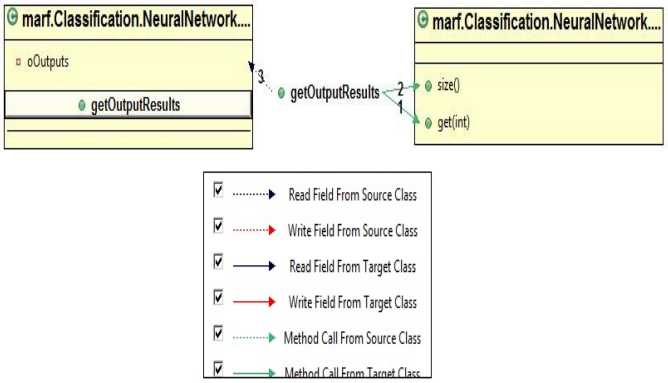

Figure 24. Code Smell Visualization

Code Smell: Issue D

Package	marf.Classification.NeuralNetwork
Source Class	NeuralNetwork
Method	setInputs(double[])
Code Smell	Feature Envy
Target Class	Layer

The left class in the diagram is NeuralNetwork and the right one is Layer. The method setInputs is in NeuralNetwork but it is more coupled with Layer.

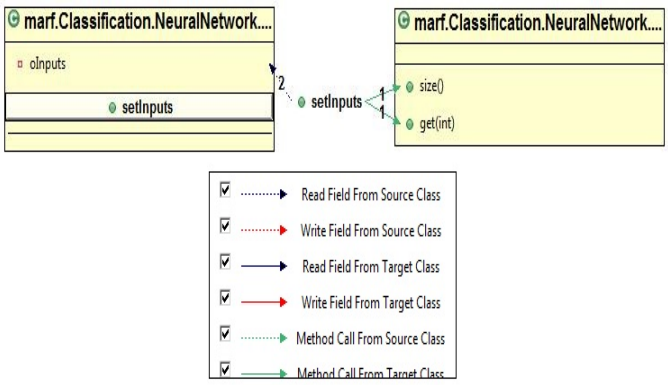

Figure 25. Code Smell Visualization

Motivation

The method is more interested in other class than the one where it is currently located. This method is in the wrong place since it is more tightly coupled to the other class than to the one where it is currently located. The solution is to create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.

Mechanics

1) First we examine all features used by the source method that are defined on the source class. We consider whether they also should be moved.
2) Second check the sub- and super-classes of the source class for other declarations of the method.
3) After that we declare the method in the target class.
4) We copy the code from the source method to the target and adjust the method to make it work in its new home.
5) Then we compile the target class.
6) Next we determine how to reference the correct target object from the source.
7) Then turn the source method into a delegating method.
8) Again compile and then test.
9) We decide whether to remove the source method or retain it as a delegating method.
10) If we remove the source method, we need to replace all the references with references to the target method.
11) At the end compile and test one more time.

3) GIPSY: Code Smell: Issue E

Package	gipsy.GIPC.DFG.DFGAnalyzer
Source Class	LucidCodeGenerator
Method	linkNode(LucidNodeItem, LucidNodeItem, int)
Code Smell	Feature Envy
Target Class	LucidNodeItem

The left class in the diagram is LucidCodeGenerator and the right one is LucidNodeItem. The method linkNode is in LucidCodeGenerator but it is more coupled with LucidNodeItem.

Please refer to Issue B for Motivations and Mechanics.

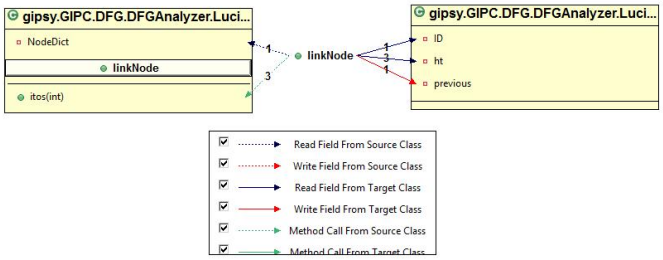

Figure 26. Code Smell Visualization

Code Smell: Issue F

Package	gipsy.GEE.multitier.GMT
Source Class	GMTWrapper
Method	handleRegDSTCrash()
Code Smell	Feature Envy
Target Class	DSTRegistration
Target Class Package	gipsy.GEE.multitier.GMT.demands

The left class in the diagram is GMTWrapper and the right one is DSTRegistration. The method handleRegDSTCrash is in GMTWrapper but it is more coupled with DSTRegistration. Please refer to Issue B for Motivations and Mechanics.

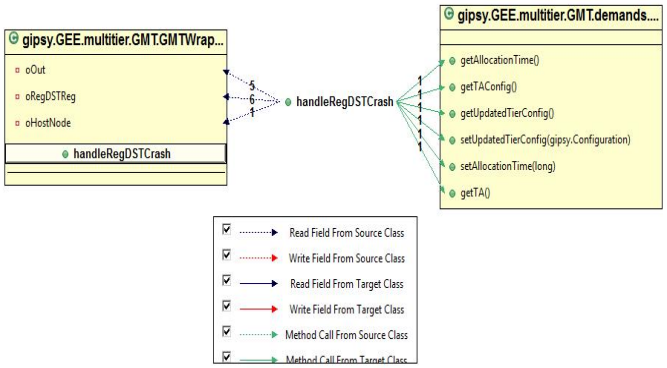

Figure 27. Code Smell Visualization

Code Smell: Issue G

Package	gipsy.GEE.multitier.GMT
Source Class	GMTInfoKeeper
Method	saveTierRegistration(TierRegistration, DSTRegistration): void
Code Smell	Type code
Code Lines	49 - 84

Motivation

We have a conditional which chooses unlike behavior reliant on the type of an object. One of the ostentatious sounding words in object terminology is polymorphism. The principle of polymorphism is that it permits to avoid writing an explicit conditional when there are objects whose behavior differs relying on their types. Therefore, if-then-else statements which switch on type strings are much less common.
Polymorphism provides many benefits. The major advantage arises when the same set of conditions occurs in many places in the program. If there is a new type to be added, one has to find and update all the conditionals. However, with subclasses one just needs to create a new subclass and add the proper methods.
So, to solve the above issue we are going to use the Replace Conditional with Polymorphism option.

Mechanics

Before beginning with Replace Conditional with Polymorphism one need to have the necessary inheritance structure. One may already have this structure from previous re-factorings. If there is no structure, it needs to be created. To create the inheritance structure there are two options: Replace Type Code with Subclasses and Replace Type Code with State/Strategy. Subclasses are the simplest option, so that can be used. If several case statements are switching on the same type code, one only needs to create one inheritance structure for that type code.
One can then work on the conditional. The code that is targeted may be a switch statement or if statement.

1) If conditional statement is part of a larger method, we separately take the conditional statement and apply Extract Method.
2) If necessary we use Move Method to place the conditional at the top of the inheritance structure.
3) We pick one of the subclasses and create a subclass method that overrides the conditional statement method.
4) After that, we copy the body of that leg of the conditional statement into the subclass method and adjust other things so that it fits.
5) We need to make some private members of the superclass protected in order to do this.
6) Next is to compile and test.
7) Then we remove the copied leg of the conditional statement.
8) We compile and test again.
9) We repeat with each leg of the conditional statement until all legs are turned into subclass methods.
10) At last we make the superclass method abstract.

Code Smell: Issue H

Package	gipsy.GEE.multitier.GMT
Old Class	GMTWrapper
New Class	GMTWrapperProduct
Method	allocateTier(String, TierIdentity, Configuration, DSTRegistration, int , GMTInfoKeeper, GMTWrapper): void; deallocateTier(String, TierIdentity, String[], GMTInfoKeeper, GMTWrapper): void
Code Smell	God Class

Motivation

God class denotes to the classes which tend to centralize the aptitude of the system. An instance of a god class does most of the work and delegates negligible details to a set of insignificant classes and using the data from other classes. A god class features high complexity and low cohesion. It disrupts the OO design principle that each class should only have one responsibility. God classes tend to be very large, which make less comprehensible. And it makes change difficult as well.
Here we have one class doing work that should be done by two. We need to create a new class and move the relevant

fields and methods from the old class to the new one. A class should be a crusty abstraction, should handle few clear and related responsibilities.
So, to solve the above issue we are going to use the Extract Class option.

Mechanics

1) We resolve how to divide the responsibilities of the class.
2) Then create a new class to shift the separated responsibilities.
3) If the responsibilities of the old class no longer tie to its name, we rename the old class.
4) We make a link from the old class to the new one.
5) We use Move Field on each field we wish to move.
6) We compile and test after each move.
7) After that we use Move Method to move methods over from old class to new class.
8) Compile and test again.
9) We review and reduce the interfaces of each class.

4) Planned Refactorings: **DMARF Refactoring: Issue A**

1) As per as the mechanics we self-encapsulate the type code.
2) We create a new class, and name it after the purpose of the type code which is INgramModel. This class is the state object.
3) Then we add subclasses of the state object, one for each type code. The subclasses are: Unigram, Bigram and Trigram
4) All the subclasses extend INgramModel.
5) After that we create abstract query in the state object INgramModel to return the type code. Subsequently, create overriding query of each state object subclass to return the correct type code. The query is:
   - readSampleData(int,double[], char[], TextLoader): int
6) Next is compiling. If the compiling does not give any error we proceed to the next step.
7) Next is to create a field in TextLoader for INgramModel object.
8) We adjust the type code query on TextLoader to delegate to INgramModel object.
9) We introduce the following methods in TextLoader:
   - getINgramEndiness(): int
   - getINgramCalculationModel(): int
   - getIPowerBase(): int
10) And then adjust the type code setting methods on TextLoader to assign an instance of the appropriate state object subclass.
11) At the end compile and test. If there is no error we are successful with the refactoring.

This refactoring affects the system design as we are introducing four new classes to the package. We have already shown the partial class diagram above. There's already a test case for this package.

Source Code for DMARF Refactoring: Issue A

```
package marf.Storage.Loaders;
public class TextLoader extends SampleLoader
{
      protected int iNgramModel = ENgramModels.BIGRAM;
      protected int iNgramCalculationModel = NGRAM_ARITHMETIC_POWER_ADD;
      protected int iNgramEndiness = NGRAM_LITTLE_ENDIAN;
      protected int iPowerBase = POWER_BASE_DECIMAL;

      public TextLoader() throws InvalidSampleFormatException{}

      public final int readSampleData(double[] padSampleData) throws
      StorageException{
            try{
                  if(iNbrChars > 0){
                        switch(this.iNgramModel){
```

Figure 28. Dmarf refactoring Issue A-Part 1

```
                        case ENgramModels.BIGRAM:{
                              for(i = 0; i < acCharBuffer.length; i += 2){
                                    switch(this.iNgramEndiness){
                                          case NGRAM_LITTLE_ENDIAN:
                                                break;
                                          case NGRAM_BIG_ENDIAN:
                                                break;
                                          default:
                                                assert false;
                                    }
                                    switch(this.iNgramCalculationModel){
                                          case NGRAM_ARITHMETIC_ADD:
                                                break;
                                          case NGRAM_ARITHMETIC_POWER_ADD:
                                                break;
                                          case NGRAM_LOGICAL_OR:
                                                break;
                                          default:
                                                assert false;
                                    }
                              }
                              break;
                        }
```

Figure 29. Dmarf refactoring Issue A-Part 2

```
                  break;
            }
            case ENgramModels.TRIGRAM:{
                  for(i = 0; i < acCharBuffer.length; i += 3){
                        switch(this.iNgramEndiness){
                              case NGRAM_LITTLE_ENDIAN:
                                    break;
                              case NGRAM_BIG_ENDIAN:
                                    break;
                              default:
                                    assert false;
                        }
                        switch(this.iNgramCalculationModel){
                              case NGRAM_ARITHMETIC_ADD:
                                    break;
                              case NGRAM_ARITHMETIC_POWER_ADD:
                                    break;
                              case NGRAM_LOGICAL_OR:
                                    break;
```

Figure 30. Dmarf refactoring Issue A-Part 3

```
                                    default:
                                          assert false;
                              }
                        }
                        break;
                  }
                  default:
                  {
                        System.err.println("WARNING: unsupported n-
gram model; falling back to unigram.");
                  }
                  case ENgramModels.UNIGRAM:{
                        break;
                  }
            }
      }
      return iNbrDataItemsRead;
}
catch(IOException e){}
    }
}
// EOF
```

Figure 31. Dmarf refactoring Issue A-Part 4

DMARF Refactoring Issue C

As the method getOutputResults(): double[] is more interested in the Layer class, we are going to move to that one.

1) We examine all features used by getOutputResults(): double[] that are defined on NeuralNetwork. We consider whether they also should be moved.
2) Second check the sub- and super-classes of NeuralNetwork class for other declarations of the method.
3) After that we declare the method in Layer class.
4) We copy the code from getOutputResults(): double[] to Layer class and adjust the method to make it work in its new home.
5) Then we compile the target class.
6) We need to decide whether to remove the source method or retain it as a delegating method.
7) If we remove the source method, we need to replace all the references with references to the target method.
8) At the end compile and test one more time.

Source Code DMARF Refactoring: Issue C GIPSY

Layer Class:

```
package marf.Classification.NeuralNetwork;
public class Layer extends BaseThread implements Serializable{

public Layer(){
          super();
     }
}
```

Figure 32. Source Code DMARF Refactoring: Issue C-Part 1

NeuralNetwork Class:

```
package marf.Classification.NeuralNetwork;
public class NeuralNetwork extends Classification
{
     private ArrayList<Layer> oLayers = new ArrayList<Layer>();
     private transient Layer oCurrentLayer;
     private Layer oInputs = new Layer();
     private Layer oOutputs = new Layer();
     public NeuralNetwork(IFeatureExtraction poFeatureExtraction)
     {
          super(poFeatureExtraction);
          this.iCurrentDumpMode = DUMP_GZIP_BINARY;
          this.strFilename = getDefaultFilename();
     }

     public double[] getOutputResults()
     {
          double[] adRet = new double[this.oOutputs.size()];
          for(int i = 0; i < this.oOutputs.size(); i++)
          {
               adRet[i] = this.oOutputs.get(i).dResult;
          }
          return adRet;
     }
}// EOF
```

Figure 33. Source Code DMARF Refactoring: Issue C-Part 2

Refactoring: Issue G

As mentioned before, before beginning with Replace Conditional with Polymorphism one needs to have the necessary inheritance structure. We already have this structure from previous re-factorings.

1) We start with GMTInfoKeeper class which is the target class.
2) First change the method saveTierRegistration: void to saveTierRegistration: TierRegistration.
3) Then return instance of TierRegistration and remove the conditional clause.
4) Again we add the method - getODGTDWTRegistration() : Map¡TierRegistration, DSTRegistration¿.
5) We make the TierRegisteration class abstract and import the necessary package.
6) We add the abstract method saveTierRegistration(DSTRegistration, GMTInfoKeeper): void
7) For DGTRegistration class we add required import declaration and add concrete message for the method - saveTierRegistration: TierRegistration
8) Again for DWTRegistration class we add required package and add concrete message for the method - saveTierRegistration: TierRegistration
9) We do the same for the DSTRegistration class as well.
10) DGTRegistration, DWTRegistration and DSTRegistration are all subclasses of TierRegisteration.

Source Code GYPSY Refactoring: Issue G

```
package gipsy.GEE.multitier.GMT;

public class GMTInfoKeeper
{
     List<NodeRegistration> oNodeRegistrations = new
ArrayList<NodeRegistration>();
     List<DSTRegistration> oDSTRegistrations = new
ArrayList<DSTRegistration>();
     Map<String, DSTRegistration> oNodeSysTARelation = new HashMap<String,
DSTRegistration>();
     Map<TierRegistration, DSTRegistration> oDGTDWTRegistration = new
HashMap<TierRegistration, DSTRegistration>();

     public synchronized void saveNodeRegistration(NodeRegistration
poRegistration, DSTRegistration poSysDST){}
```

Figure 34. Source Code DMARF Refactoring: Issue G-Part 1

```
     public void saveTierRegistration(TierRegistration poRegistration,
DSTRegistration poDataDST){
          if(poRegistration instanceof DSTRegistration){
               DSTRegistration oNewDSTReg = (DSTRegistration) poRegistration;
               synchronized(this.oDSTRegistrations){
                    for(int i = 0; i<this.oDSTRegistrations.size(); i++){
                         DSTRegistration oDSTReg =
this.oDSTRegistrations.get(i);

     if(oDSTReg.getNodeID().equals(oNewDSTReg.getNodeID())
                                        &&
oDSTReg.getTierID().equals(oNewDSTReg.getTierID()))
                              {
                                   this.oDSTRegistrations.set(i, oNewDSTReg);
                                   return;
                              }
                    }
```

Figure 35. Source Code DMARF Refactoring: Issue G-Part 2

```
                    this.oDSTRegistrations.add((DSTRegistration)
poRegistration);
                }
            }
            else if(poRegistration instanceof DGTRegistration ||
                    poRegistration instanceof DWTRegistration)
            {
                if(poDataDST != null){
                    synchronized(poDataDST)
                    {
                        int iActiveConnectCount =
poDataDST.getActiveConnectionCount() + 1;
                    poDataDST.setActiveConnectionCount(iActiveConnectCount);
                    }
                }
                synchronized(this.oDGTDWTRegistration)
                {
                    this.oDGTDWTRegistration.put(poRegistration, poDataDST);
                }
            }
        }
}
```

Figure 36. Source Code DMARF Refactoring: Issue G-Part 3

GYPSY Refactoring: Issue H

As mentioned above we are going to break the God Class in following steps:

1) We resolve how to divide the responsibilities of the GMTWrapper class.
2) Then create a new class GMTWrapperProduct to shift the separated responsibilities.
3) The responsibilities of the old class still tie to its name, so there is no need to rename the old class.
4) We make a link from GMTWrapper to GMTWrapperProduct. We create a GMTWrapperProduct instance in the GMTWrapper class.
5) We use Move Field on each field we wish to move.
6) We compile and test after each move.
7) After that we use Move Method to move methods over from GMTWrapper class to GMTWrapperProduct class. First we are going to move the method: allocateTier: void
8) Then we compile and test.
9) If everything is fine, we are going to move the method: deallocateTier: void
10) Compile and test again.
11) We check if we need to move anything else to make the refactoring complete.

This refactoring affects the system design as we are introducing a new class to the package. Again, there's no test case for this. We are going to make a test unit for testing this change.

Source Code GIPSY Refactoring: Issue H

```
package gipsy.GEE.multitier.GMT;
public class GMTWrapper extends GenericTierWrapper
{
    public GMTInfoKeeper oInfoKeeper = new GMTInfoKeeper();
    private DSTRegistration oRegDSTReg = null;
    private GIPSYNode oHostNode = null;
    public GMTWrapper(){}

    public GMTWrapper(Configuration poGMTConfig){
        this.oConfiguration = poGMTConfig;
        this.oOut = System.out;
        this.oErr = System.err;
        this.strTierID = this.oConfiguration.getProperty(GMT_TIERID);
        this.oHostNode =
(GIPSYNode)this.oConfiguration.getObjectProperty(GMT_NODE);
    }

    public void run(){}

    public void startTier() throws MultiTierException{}
```

Figure 37. Source Code GIPSY Refactoring: Issue H-Part 1

```
    public void allocateTier(String pstrNodeID, TierIdentity poTierIdentity,
            Configuration poTierConfig, DSTRegistration poDataDSTReg, int
piNumOfInstances) throws MultiTierException
    {
        DSTRegistration oSysTAReg =
this.oInfoKeeper.getNodeSysDST(pstrNodeID);
        ITransportAgent oSysTA = oSysTAReg.getTA();

        TierAllocationRequest oRequest = new
TierAllocationRequest(pstrNodeID, poTierIdentity, poTierConfig,
piNumOfInstances);
        String strTierID =
poTierConfig.getProperty(IMultiTierWrapper.WRAPPER_TIER_ID);

        boolean bIsReallocation = false;
        TierAllocationResult oResult = null;
        TierRegistration[] aoRegistrations = oResult.getRegistrations();
    }
```

Figure 38. Source Code GIPSY Refactoring: Issue H-Part 2

```
    public void deallocateTier(String pstrNodeID, TierIdentity poTierIdentity,
String[] pastrTierIDs) throws MultiTierException
    {
        DSTRegistration oSysTAReg =
this.oInfoKeeper.getNodeSysDST(pstrNodeID);

        ITransportAgent oSysTA = oSysTAReg.getTA();

        TierDeallocationRequest oRequest = new
TierDeallocationRequest(pstrNodeID, poTierIdentity, pastrTierIDs);

        TierDeallocationResult oResult = null;
    }
```

Figure 39. Source Code GIPSY Refactoring: Issue H-Part 3

```
    private void handleRegDSTCrash(){}
    public GMTInfoKeeper getInfoKeeper(){}
    public void stopTier() throws MultiTierException{}
    public void setOut(PrintStream poOut){}
    public void setErr(PrintStream poErr){}
    private DSTRegistration getAnAvailableDST(){}
    private DSTRegistration handleNonRegDSTCrash(DSTRegistration
    poCrashedDSTReg){}
}//EOF
```

Figure 40. Source Code GIPSY Refactoring: Issue H-Part 4

5) JDeodorant Tool: We used JDeodorant tool to find bad smells present in DMARF and GIPSY [27]. Steps to install and use JDeodorant tool, and detect Bad smells that we used are illustrated below:

Installation:

1) We go to eclipse Eclipse Marketplace (**Help ->Install Eclipse Marketplace**) to install the eclipse JDeodorant plugin.
2) Search **JDeodorant** in the marketplace and click on confirm button.
3) Accept license while installing plug-in.
4) In case of warning message, press OK to continue and restart eclipse after installation.
5) Add following configuration parameters to **eclipse.ini**
   -vmargs
   -Xms128m
   -Xmx2048m
   -XX:PermSize=128m

Usage:

JDeodorant is used to find following types of bad smells:

1) God Class
2) Long Method
3) Type Checking
4) Feature Envy

Among above four we found out Type Checking and Feature Envy. Steps used to discover these smells which are are given below in proper manner:

Type Checking

Steps for Type Checking are as following:

1) From Package Explorer View (**Window ->Show View ->Package Explorer**) and Checking View (**Bad Smells ->Type Checking**).
2) Select the source code of the project and click Identify Bad Smell button. (Symbol looks like I on the top right corner of the console).
3) In table generated by JDeodorant, we see all the possible refactoring.
4) We double clicked on particular row to open corresponding class where there could be the possibility of refactoring; we have seen relevant code highlighted.

Feature Envy Steps for Type Checking are as following:

1) From Package Explorer View (**Window ->Show View ->Package Explorer**) and Checking View (**Bad Smells ->Feature Envy**).
2) Select the source code of the project and click Identify Bad Smell button. (Symbol looks like I on the top right corner of the console).
3) In table generated by JDeodorant, we see all the possible refactoring.
4) We double clicked on particular row to open corresponding class where there could be the possibility of refactoring; we have seen relevant code highlighted.

VI. Identification of Design Patterns

We have identified some of the design patterns by using the tool Design Pattern Recognizer. To find the rest we have gone through the design and source codes. We have also used ObjectAid to find out the relations between the classes.

A. DMARF

Composite (Structural GoF pattern)

The composite pattern describes that a group of objects is to be treated in the same way as a single instance of an object. The intent of a composite is to ”compose” objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern helps to treat individual objects and compositions uniformly.

Related classes:-

Component: ASSLEVENTCATCHER

Component is the abstraction for all components and declares the interface for objects in the composition.

Composite: ASSLEVENT

Composite represents a composite Component and implements Component method by delegating them to its children.

Leaves: ASSLFLUENT, ASSLRECOVERY-PROTOCOL

Leaf represents leaf objects in the composition and implements all Component methods.

Relations:

Role A	Relation	Role B
Composite	aggregate	Component
Component	is parent of	Composite
Component	is parent of	Leaf

Class Diagram:

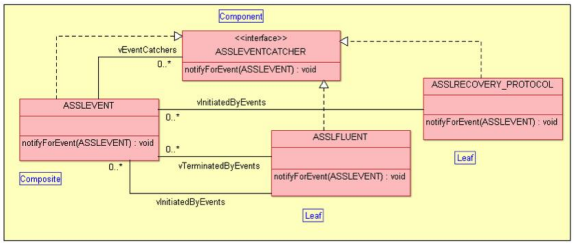

Figure 41. Composite Pattern Diagram

Composite Pattern Source Code

Component (Interface): ASSLEVENTCATCHER

```
package marf.net.assl.generatedbyassl.as;

public interface ASSLEVENTCATCHER
{
        public void notifyForEvent ( ASSLEVENT poEvent );
}
```

Composite (Class): ASSLEVENT

```
package marf.net.assl.generatedbyassl.as;

public class ASSLEVENT extends Thread
        implements  ASSLEVENTCATCHER, ASSLMESSAGECATCHER
{
        public synchronized void notifyForEvent ( ASSLEVENT poEvent ){}
}
```

Figure 42. Composite Pattern Source Code Part-1

Leaf (Class): ASSLFLUENT

```
package marf.net.assl.generatedbyassl.as;

public class ASSLFLUENT extends Thread implements  ASSLEVENTCATCHER
{
      public synchronized void notifyForEvent ( ASSLEVENT poEvent ){}
}
```

Leaf (Class): ASSLRECOVERY_PROTOCOL

```
package marf.net.assl.generatedbyassl.as;

public class ASSLRECOVERY_PROTOCOL implements  ASSLEVENTCATCHER
{
      public synchronized void notifyForEvent ( ASSLEVENT poEvent ){}
}
```

Figure 43. Composite Pattern Source Code Part-2

Facade (Structural GoF pattern)

A facade is an object that provides a simplified interface to a larger body of code. It is used when one wants an easier or simpler interface to an underlying implementation object.
Related classes:-
Client: Complier
The object is using the Facade to access resources from the Subsystems.
Facade: Parser
The facade class abstracts Subsystems from the rest of the application.
Subsystem: LexicalAnalyzer, SymbolTable
Relations:

Role A	Relation	Role B
Facade	aggregate	Subsystem
Facade	associate with	Subsystem
Client	associate with	Facade

Class Diagram:

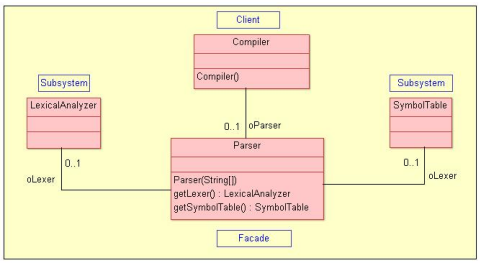

Figure 44. Facade Class Diagram

Facade Pattern Source Code

Source Code

Client (Class): Complier

```
package marf.nlp.Parsing;

public class Compiler
{
      public Compiler(){}
}
```

Façade (Class): Parser

```
package marf.nlp.Parsing;

public class Parser
{
      public Parser(String[] argv) throws CompilerError{}

      public LexicalAnalyzer getLexer(){}

      public static SymbolTable getSymbolTable(){}
}
```

Subsystem (Class): LexicalAnalyzer

```
package marf.nlp.Parsing;

public class LexicalAnalyzer extends GenericLexicalAnalyzer{}
```

Subsystem (Class): SymbolTable

```
package marf.nlp.Parsing;

public class SymbolTable{}
```

Figure 45. Facade Design Pattern Source Code

State (Behavioral GoF pattern) State pattern is used to encapsulate varying behavior for the same routine based on an object's state object. This can be a cleaner way for an object to change its behavior at runtime without resorting to large monolithic conditional statements.
Related Classes:
Context: MaxProbabilityClassifier
State: StatisticalEstimator
Concrete_State: SLI, GLI
Relations:

Role A	Relation	Role B
State	is parent of	Concrete_state
Context	associate with	State
Context	aggregate	State

Class Diagram State Design Pattern Source Code

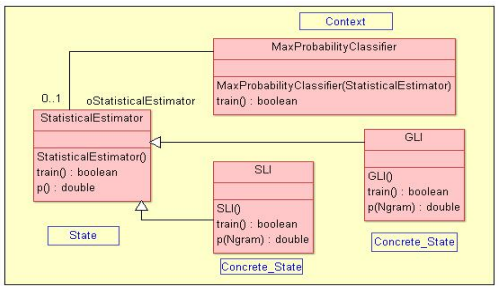

Figure 46. State Class Diagram

Context (Class): MaxProbabilityClassifier

```
package marf.Classification.Stochastic;
public class MaxProbabilityClassifier extends Stochastic
{
	protected StatisticalEstimator oStatisticalEstimator = null;
	protected Vector oAvailLanguages = null;
	public MaxProbabilityClassifier(StatisticalEstimator
poStatisticalEstimator){}
	public boolean train() throws ClassificationException{}
}
```

State (Abstract Class): StatisticalEstimator

```
package marf.Stats.StatisticalEstimators;
public abstract class StatisticalEstimator extends StorageManager
implements IStatisticalEstimator
{
	public StatisticalEstimator(){}
	public boolean train(){}
	public final double p(){}
}
```

Concrete_State (Class): SLI

```
package marf.Stats.StatisticalEstimators;
public class SLI extends StatisticalEstimator
{
	public SLI(){}
	public boolean train(){}
	public double p(Ngram poNgram){}
}
```

Concrete_State (Class): GLI

```
package marf.Stats.StatisticalEstimators;
public class GLI extends StatisticalEstimator
{
	public GLI(){}
	public boolean train(){}
	public double p(Ngram poNgram){}
}
```

Figure 47. State Design Pattern Source Code

Factory (Creational GoF pattern) The factory pattern returns a specific class based on the input.
Related Classes:
Simple Factory: MARF
This class returns a specific class depending on the input.
Child Interface: IClassification, IFeatureExtraction, IPreprocessing, ISampleLoader.

Child interfaces are working more like abstract factories which have concrete factories which return different types of classes depending on the inputs.

Class Diagram

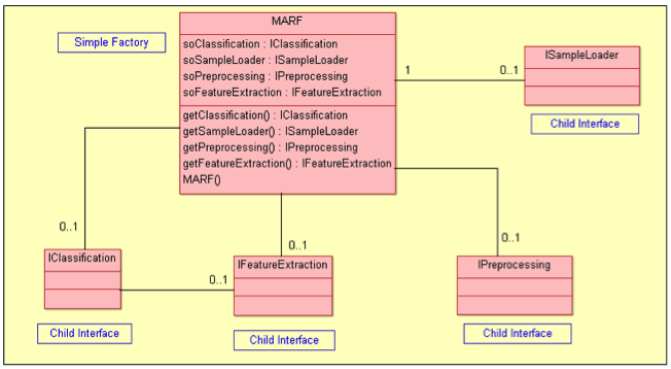

Figure 48. Factory Class Diagram

Factory Design Pattern Source Code

Simple Factory (Class): MARF

```
package marf;
public class MARF
{
	private static IClassification    soClassification    = null;

	private static ISampleLoader      soSampleLoader      = null;

	private static IPreprocessing     soPreprocessing     = null;

	private static IFeatureExtraction soFeatureExtraction = null;

	public static synchronized final IClassification
getClassification(){}

	public static synchronized final ISampleLoader
getSampleLoader(){}

	public static synchronized final IPreprocessing
getPreprocessing(){}

	public static synchronized final IFeatureExtraction
getFeatureExtraction(){}

	private MARF(){}
}
```

Child Interface: IClassification.

```
package marf.Classification;
public interface IClassification{}
```

Child Interface: IFeatureExtraction.

```
package marf.FeatureExtraction;
public interface IFeatureExtraction{}
```

Child Interface: IPreprocessing.

```
package marf.Preprocessing;

public interface IPreprocessing extends Cloneable{}
```

Child Interface: ISampleLoader.

```
package marf.Storage;
public interface ISampleLoader{}
```

Figure 49. Factory Design Pattern Source Code

B. GIPSY

Strategy (Behavioral GoF pattern)

A class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors. These factors are not known for each case until run-time, and may require radically different validation to be performed.

The validation strategies, encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.
Related Classes:
Context: IdentifierContext
Strategy: IValueStore
Concrete_Strategy: ValueHouse, DemandHashtable
Relations:

Role A	Relation	Role B
Strategy	is parent of	Concrete_strategy
Context	associate with	Strategy
Context	aggregate	Strategy

Class Diagram Strategy Design Pattern Source Code

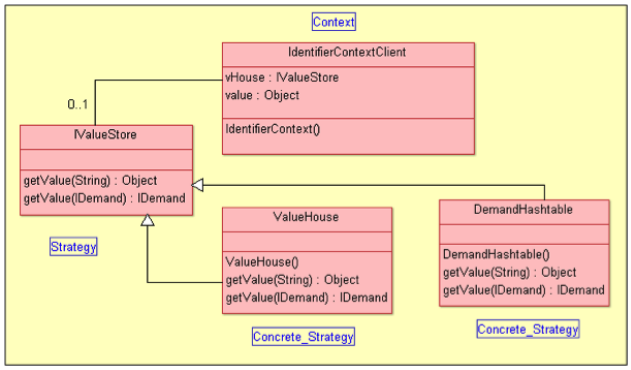

Figure 50. Strategy Class Diagram

Context (Class): IdentifierContextClient

```
package gipsy.GEE.IDP.DemandGenerator.rmi;
public class IdentifierContext extends IdentifierContextClient implements
CONFIG
{
	private Object oValue;
	public IdentifierContext(){}
}
```

Strategy (Interface): IValueStore

```
package gipsy.GEE.IVW.Warehouse;
public interface IValueStore
{
	@Deprecated
	public Object getValue(String pstrDemand) throws GEEException;
	public IDemand getValue(IDemand poDemand)
	throws GEEException, RemoteException;
}
```

Concrete_ Strategy (Class): ValueHouse

```
package gipsy.GEE.IVW.Warehouse;
public class ValueHouse extends UnicastRemoteObject implements IValueStore,
Remote
{
	public ValueHouse() throws RemoteException{}
	public Object getValue(String pstrDimension){}
	@Override
	public IDemand getValue(IDemand poDemand) throws GEEException,
RemoteException
	{}
}
```

Concrete_ Strategy (Class): DemandHashtable

```
package gipsy.GEE.IDP.DemandGenerator.threaded;
public class DemandHashtable implements IDemandList, IValueStore
{
	public DemandHashtable(){}
	public synchronized Object getValue(String pstrDemand) throws
GEEException{}
	@Override
	public IDemand getValue(IDemand poDemand) throws GEEException{}
}
```

Figure 51. Strategy Design Pattern Source Code

Singleton (Creational GoF pattern)

Singleton pattern is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects.

Related Classes:

Singleton: TAFactory

Implementation of a singleton pattern must satisfy the single instance and global access principles. It requires a mechanism to access the singleton class member without creating a class object and a mechanism to persist the value of class members among class objects. The singleton pattern is implemented by creating a class with a method that creates a new instance of the class if one does not exist. If an instance already exists, it simply returns a reference to that object. To make sure that the object cannot be instantiated any other way, the constructor is made private.

Class Diagram

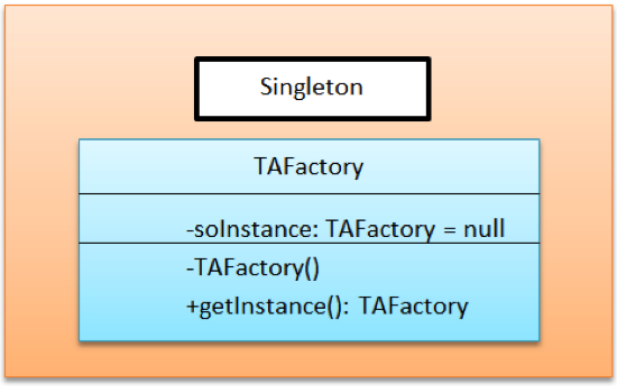

Figure 52. Singleton Class Diagram

Singleton Design Pattern Source Code

Singleton (Class): TAFactory

```
package gipsy.GEE.multitier.DST;

public class TAFactory
{
	private static TAFactory soInstance = null;

	private TAFactory(){}

	public synchronized static TAFactory getInstance(){}
}
```

Figure 53. Singleton Design Pattern Source Code

Observer (Behavioral GoF pattern)

The Observer defines a one-to-many relationship so that when one object changes state, the others are notified and updated automatically. The Subject prompts the Observer objects to do their thing. Each Observer can call back to the Subject as needed.

Related Classes:

Subject: DGTFactory

Observer: DGTWrapper

Concrete_Observer: LocalGEERPool, LocalDemandStore

Relations:

Role A	Relation	Role B
Observer	associate with	Concrete_Observer
Concrete_Observer	associate with	Subject
Observer	associate with	Subject

Class Diagram

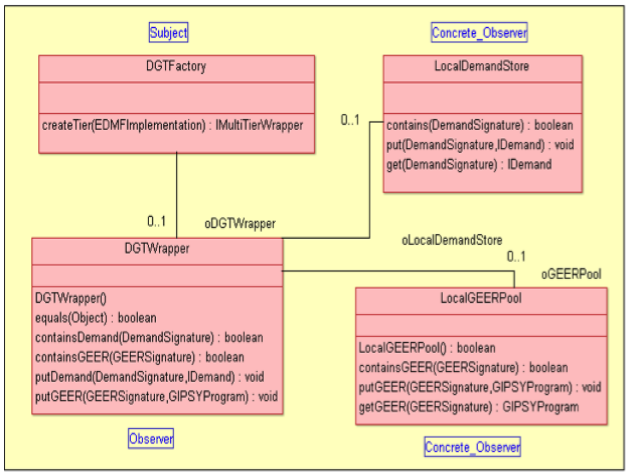

Figure 54. Observer Design Pattern

Observer Design Pattern Source Code

Subject (Class): DGTFactory

```
package gipsy.GEE.multitier.DGT;
public class DGTFactory extends TierFactory
{
	@Override
	public IMultiTierWrapper createTier(EDMFImplementation poDMFImp)
	throws MultiTierException{}
}
```

Observer (Class): DGTWrapper

```
package gipsy.GEE.multitier.DGT;
public class DGTWrapper extends GenericTierWrapper
{
public DGTWrapper() throws GEEException{}
public boolean equals(Object poDGT){}
public boolean containsDemand(DemandSignature poDemandSignature){}
public boolean containsGEER(GEERSignature poGEERSignature){}
public void putDemand(DemandSignature poDemandSignature, IDemand poDemand){}
public void putGEER(GEERSignature poGEERSignature, GIPSYProgram
poGIPSYProgram){}
}
```

Concrete_ Observer (Class): LocalGEERPool

```
package gipsy.interfaces;
public class LocalGEERPool
{
	public LocalGEERPool(){}
	public boolean contains(GEERSignature poGEERSignature){}
	public void put(GEERSignature poGEERSignature, GIPSYProgram poGEER){}
	public GIPSYProgram get(GEERSignature poGEERSignature){}
}
```

Concrete_ Observer (Class): LocalDemandStore

```
package gipsy.interfaces;
public class LocalDemandStore
{
public synchronized boolean contains(DemandSignature poDemandSignature){}
public synchronized void put(DemandSignature poDemandSignature, IDemand
poResult){}
public synchronized IDemand get(DemandSignature poDemandSignature){}
}
```

Figure 55. Observer Design Pattern Source Code

C. Design Pattern Recognizer Tool

We used Design Pattern Recognizer tool to detect most common design patterns present in DMARF and GIPSY [32]. Steps to install and use Recognizer tool, and detect pattern that we used are illustrated below:

Installation:

1) We go to Help and then Click on Install New Software to install the plug-in for this tool.This is an essential step to find design pattern.
2) Put the proper link under "Work with" and Click on 'Add' button. The link is http://lubes.yweb.sk/projects/dprecognizer/update/
3) Click Next button and eclipse will install Design Pattern Recognizer.
4) Accept license while installing plug-in.
5) In case of warning message, press OK to continue and restart eclipse after installation.

Usage:

1) Go to **Window** ->**show view** ->others and select **Design Pattern Recognizer** from Recognizer.
2) Select Project in Design Pattern Recognizer tab.
3) It will tell you five types of patterns present in selected project that are **COMPOSITE**, **FACADE**, **OBSERVER**, **SINGLETON** and **STRATEGY**.

VII. Implementation

A. Refactoring Changesets and Diffs

1) DMARF: **Issue A:** The Changesets and diffs for the DMARF-Issue A are given below which are very important because they are helpful to identity extact point of change and the reasons for change.

```
package marf.Storage.Loaders
TextLoader 1.1.1.1
Line 24
 * $Id: TextLoader.java,v 1.1.1.1 2014/08/14 02:46:59 mu_kumar Exp $
 Line 37
protected int iNgramModel = ENgramModels.BIGRAM; //NLP.getNgramModel();
Line 302
return "$Revision: 1.1.1.1 $";
Lines 121-240
switch(this.iNgramModel)
{
		case ENgramModels.BIGRAM:
		{
			switch(this.iNgramEndiness)
			{
				case NGRAM_LITTLE_ENDIAN:
					break;
				case NGRAM_BIG_ENDIAN:
					break;
				default:
			}
			switch(this.iNgramCalculationModel)
			{
				case NGRAM_ARITHMETIC_ADD:
					break;
				case NGRAM_ARITHMETIC_POWER_ADD:
					break;
				case NGRAM_LOGICAL_OR:
					break;
				default:
			}}
	break;
```

Figure 56. Changesets and Diffs for Issue A- Part 1

```
case ENgramModels.TRIGRAM:
{
              switch(this.iNgramEndiness)
              {
                     case NGRAM_LITTLE_ENDIAN:
                            break;
                     case NGRAM_BIG_ENDIAN:
                            break;
                     default:
              }

              switch(this.iNgramCalculationModel)
              {
                     case NGRAM_ARITHMETIC_ADD:
                            break;
                     case NGRAM_ARITHMETIC_POWER_ADD:
                            break;
                     case NGRAM_LOGICAL_OR:
                            break;
                     default:
              }
       }
       break;
}
default:
{
}
case ENgramModels.UNIGRAM:
{
       Arrays.copy(padSampleData, acCharBuffer);
       iNbrDataItemsRead = padSampleData.length;
       break;
```

Figure 57. Changesets and Diffs for Issue A- Part 2

```
.
default:
{
}
case ENgramModels.UNIGRAM:
{
      Arrays.copy(padSampleData, acCharBuffer);
      iNbrDataItemsRead = padSampleData.length;
      break;
}
}

TextLoader 1.1.1.2
Line 24
 * $Id: TextLoader.java,v 1.1.1.2 2014/08/22 01:00:10 sale_ahm Exp $

Line 37
protected TextLoader_INgramModel iNgramModel = new TextLoader_Bigram();
//NLP.getNgramModel();

Lines 186-218
public void setTextLoader_INgramModel(int iNgramModel) {
            switch (iNgramModel) {
            case ENgramModels.BIGRAM:
                  this.iNgramModel = new TextLoader_Bigram();
                  break;
            case ENgramModels.TRIGRAM:
                  this.iNgramModel = new TextLoader_Trigram();
                  break;
            case ENgramModels.UNIGRAM:
                  this.iNgramModel = new TextLoader_Unigram();
                  break;
            default:
```

Figure 58. Changesets and Diffs for Issue A- Part 3

```
                  this.iNgramModel = null;
                  break;
            }
}
public int getTextLoader_INgramModel() {
             return iNgramModel.getTextLoader_INgramModel();}
public int getINgramEndiness() {
             return iNgramEndiness;}
public int getINgramCalculationModel() {
             return iNgramCalculationModel;
}
public int getIPowerBase() {
             return iPowerBase;
}
Line 184
return "$Revision: 1.1.1.2 $";
Lines 121-122
      iNbrDataItemsRead = iNgramModel.readSampleData(
                                    iNbrDataItemsRead, padSampleData,
acCharBuffer, this);
TextLoader 1.1.1.3
Line 24
 * $Id: TextLoader.java,v 1.1.1.3 2014/08/23 22:21:57 sale_ahm Exp $
Line 186
return "$Revision: 1.1.1.3 $";
Line 121-122
//Change by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
//Refactoring code by removing Switch cases y strategy pattern
```

Figure 59. Changesets and Diffs for Issue A- Part 4

```
package marf.Storage.Loaders
TextLoader_INgramModel 1.1.2.1
New class added to the package.
TextLoader_INgramModel 1.1.2.2
Lines 1-2//Class added by Aditya Dewal, Ashish Arora, Kanwaldeep Singh,
Lovepreet Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
//Class added as Strategy pattern Class for Unigram, Trigram,
Bigram as INgramModel
package marf.Storage.Loaders
TextLoader_Bigram 1.1.2.1
New class added to the package.
TextLoader_Bigram 1.1.2.2
Lines 6-7//Class added by Aditya Dewal, Ashish Arora, Kanwaldeep Singh,
Lovepreet Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
//Refactoring code by Adding Concrete Strategy pattern Class for
Bigram
package marf.Storage.Loaders
TextLoader_Trigram 1.1.2.1
New class added to the package.
TextLoader_Trigram 1.1.2.2
Lines 6-7//Code added by Aditya Dewal, Ashish Arora, Kanwaldeep Singh,
Lovepreet Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
//Refactoring code by Adding Concrete Strategy pattern Class for
Trigram
package marf.Storage.Loaders
TextLoader_Unigram 1.1.2.1
New class added to the package.
TextLoader_Unigram 1.1.2.2
Lines 6-7//Code added by Aditya Dewal, Ashish Arora, Kanwaldeep Singh,
Lovepreet Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
//Refactoring code by Adding Concrete Strategy pattern Class for Unigram
```

Figure 60. Changesets and Diffs for Issue A- Part 5

Issue C:

```
package marf.Classification.NeuralNetwork
NeuralNetwork 1.1.1.1
Line 38
 * $Id: NeuralNetwork.java,v 1.1.1.1 2014/08/14 02:46:53 mu_kumar Exp $
Line 1390
return "$Revision: 1.1.1.1 $";
Lines 874-888
/**
       * Gets outputs of a neural network run.
       * @return array of doubles read off the output layer's neurons
       */
      public double[] getOutputResults()
      {
            double[] adRet = new double[this.oOutputs.size()];
            for(int i = 0; i < this.oOutputs.size(); i++)
            {
                  adRet[i] = this.oOutputs.get(i).dResult;
            }
            return adRet;
}
NeuralNetwork 1.1.1.2
Line 38
 * $Id: NeuralNetwork.java,v 1.1.1.2 2014/08/22 23:25:08 sale_ahm Exp $
Line 1376
return "$Revision: 1.1.1.2 $";
Line 43
 * @version $Revision: 1.1.1.2 $
NeuralNetwork 1.1.1.3
Line 38
 * $Id: NeuralNetwork.java,v 1.1.1.3 2014/08/23 22:30:33 sale_ahm Exp $
Line 43
 * @version $Revision: 1.1.1.3 $
Line 1378
return "$Revision: 1.1.1.3 $";
package marf.Classification.NeuralNetwork
Layer 1.1.1.1
```

Figure 61. Changesets and Diffs for Issue C- Part 1

```
Line 18
 * $Id: Layer.java,v 1.1.1.1 2014/08/14 02:46:53 mu_kumar Exp $
Line 171
return "$Revision: 1.1.1.1 $";
Layer 1.1.1.2
Line 18
 * $Id: Layer.java,v 1.1.1.2 2014/08/22 23:25:08 sale_ahm Exp $
Line 171
return "$Revision: 1.1.1.2 $";
Line 173-184
/**
       * Gets outputs of a neural network run.
       * @return  array of doubles read off the output layer's neurons
*/
public double[] getOutputResults() {
      double[] adRet = new double[size()];
      for (int i = 0; i < size(); i++) {
            adRet[i] = get(i).dResult;
      }
      return adRet;
}
Line 22
 * @version $Revision: 1.1.1.2 $
Layer 1.1.1.3
Line 18
 * $Id: Layer.java,v 1.1.1.3 2014/08/23 22:30:33 sale_ahm Exp $
Line 22
 * @version $Revision: 1.1.1.3 $
Line 171
return "$Revision: 1.1.1.3 $";
```

Figure 62. Changesets and Diffs for Issue C- Part 2

2) GIPSY: **Issue G:**

```
package gipsy.GEE.multitier.GMT

GMTInfoKeeper.java 1.1.1.1

Line 25
* @version $Id: GMTInfoKeeper.java,v 1.1.1.1 2014/08/14 02:11:02 mu_kumar Exp $

Lines 67-104
public void saveTierRegistration(TierRegistration poRegistration,
DSTRegistration poDataDST)
        {
                if(poRegistration instanceof DSTRegistration)
                {
                        DSTRegistration oNewDSTReg = (DSTRegistration) poRegistration;
                        synchronized(this.oDSTRegistrations)
                        {
                                for(int i = 0; i<this.oDSTRegistrations.size(); i++)
                                {
                                        DSTRegistration oDSTReg =
this.oDSTRegistrations.get(i);
       if(oDSTReg.getNodeID().equals(oNewDSTReg.getNodeID())
                                                        &&
oDSTReg.getTierID().equals(oNewDSTReg.getTierID()))
                                {
                                        this.oDSTRegistrations.set(i, oNewDSTReg);
                                        return;
                                }
                        }
                        this.oDSTRegistrations.add((DSTRegistration)
poRegistration);
                }
                else if(poRegistration instanceof DGTRegistration ||
                                poRegistration instanceof DWTRegistration)
                {
                        if(poDataDST != null)
                        {
                                synchronized(poDataDST)
                                {
                                        int iActiveConnectCount =
poDataDST.getActiveConnectionCount() + 1;
       poDataDST.setActiveConnectionCount(iActiveConnectCount);
                                }

                        }
                        synchronized(this.oDGTDWTRegistration)
                        {
                                this.oDGTDWTRegistration.put(poRegistration, poDataDST);
                        }
                }
```

Figure 63. Changesets and Diffs for Issue G- Part 1

```
GMTInfoKeeper.java 1.1.1.2

Line 25
* @version $Id: GMTInfoKeeper.java,v 1.1.1.2 2014/08/22 03:30:46 sale_ahm Exp $

Lines 67 - 70
public TierRegistration saveTierRegistration(TierRegistration poRegistration,
DSTRegistration poDataDST){
                return poRegistration.saveTierRegistration(poDataDST, this);
        }

Lines 318 - 320
public Map<TierRegistration, DSTRegistration> getODGTDWTRegistration() {
                return oDGTDWTRegistration;
        }

GMTInfoKeeper.java 1.1.1.3

Line 25
 * @version $Id: GMTInfoKeeper.java,v 1.1.1.3 2014/08/23 23:17:57 sale_ahm Exp $

Lines 69-70
      //Code change by Aditya Dewal, Aashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel.
      // If conditions identified to be removed from polymorphism to remove code
smells.

Lines 319-320
      //Code change by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel.
      //Function Map added to support polymorphism.

package gipsy.GEE.multitier.GMT.demands

DGTRegistration 1.1.1.1
Line 14
 * @version $Id: DGTRegistration.java,v 1.1.1.1 2014/08/14 02:11:02 mu_kumar Exp
$

DGTRegistration 1.1.1.2

Line 9
import gipsy.GEE.multitier.GMT.GMTInfoKeeper;

Line 14
 * @version $Id: DGTRegistration.java,v 1.1.1.2 2014/08/22 03:30:46 sale_ahm Exp
$

Line 31-44
public TierRegistration saveTierRegistration(DSTRegistration poDataDST,
                GMTInfoKeeper gMTInfoKeeper) {
                if (poDataDST != null) {
                        synchronized (poDataDST) {
```

Figure 64. Changesets and Diffs for Issue G- Part 2

```
                                int iActiveConnectCount =
poDataDST.getActiveConnectionCount() + 1;
                                poDataDST.setActiveConnectionCount(iActiveConnectCount);
                        }
                synchronized (gMTInfoKeeper.getODGTDWTRegistration()) {
                        gMTInfoKeeper.getODGTDWTRegistration().put(this, poDataDST);
                }
                return null;
        }

DGTRegistration 1.1.1.3

Lines 14-16
 * @version $Id: DGTRegistration.java,v 1.1.1.3 2014/08/23 23:17:57 sale_ahm Exp
$
 * Code Refactored by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
 * Function TierRegistration added as part of Polymorphism

package gipsy.GEE.multitier.GMT.demands

DSTRegistration 1.1.1.1
Line 14
 * @version $Id: DSTRegistration.java,v 1.1.1.1 2014/08/14 02:11:02 mu_kumar Exp
$


DSTRegistration 1.1.1.2

Line 10
import gipsy.GEE.multitier.GMT.GMTInfoKeeper;

Line 15
 * @version $Id: DSTRegistration.java,v 1.1.1.2 2014/08/22 03:30:46 sale_ahm Exp
$

Line 100-104
public TierRegistration saveTierRegistration(DSTRegistration poDataDST,
                GMTInfoKeeper gMTInfoKeeper) {
                return null;
        }

DSTRegistration 1.1.1.3

 * @version $Id: DSTRegistration.java,v 1.1.1.3 2014/08/23 23:17:57 sale_ahm Exp
$
 * Code Refactored by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
 * Function TierRegistration added as part of Polymorphism

 package gipsy.GEE.multitier.GMT.demands
 DWTRegistration 1.1.1.1
 Line 14
```

Figure 65. Changesets and Diffs for Issue G- Part 3

```
  * @version $Id: DWTRegistration.java,v 1.1.1.1 2014/08/14 02:11:02 mu_kumar
Exp $

DWTRegistration 1.1.1.2

Line 9
 import gipsy.GEE.multitier.GMT.GMTInfoKeeper;

Line 14
  * @version $Id: DWTRegistration.java,v 1.1.1.2 2014/08/22 03:30:46 sale_ahm
Exp $

Lines 31-49
public TierRegistration saveTierRegistration(DSTRegistration poRegistration,
                GMTInfoKeeper gMTInfoKeeper) {
                DSTRegistration oNewDSTReg = (DSTRegistration) poRegistration;
                synchronized (gMTInfoKeeper.getDSTRegistrations()) {
                        for (int i = 0; i <
gMTInfoKeeper.getDSTRegistrations().size(); i++) {
                                DSTRegistration oDSTReg =
gMTInfoKeeper.getDSTRegistrations()
                                                .get(i);
                                if (oDSTReg.getNodeID().equals(oNewDSTReg.getNodeID())
                                                &&
oDSTReg.getTierID().equals(oNewDSTReg.getTierID())) {
                                        gMTInfoKeeper.getDSTRegistrations().set(i,
oNewDSTReg);
                                        return null;
                                }
                        }
                        gMTInfoKeeper.getDSTRegistrations().add(
                                        (DSTRegistration) poRegistration);
                }
                return null;
        }

DWTRegistration 1.1.1.3
 * @version $Id: DWTRegistration.java,v 1.1.1.3 2014/08/23 23:17:57 sale_ahm Exp
$
 * * Code Refactored by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
 * Function TierRegistration added as part of Polymorphism

package gipsy.GEE.multitier.GMT.demands

TierRegistration 1.1.1.1

Line 12
 * @version $Id: TierRegistration.java,v 1.1.1.1 2014/08/14 02:11:02 mu_kumar
Exp $

TierRegistration 1.1.1.2
Line 7
import gipsy.GEE.multitier.GMT.GMTInfoKeeper;
```

Figure 66. Changesets and Diffs for Issue G- Part 4

```
Line 13
 * @version $Id: TierRegistration.java,v 1.1.1.2 2014/08/22 03:30:46 sale_ahm
Exp $


Lines 53-55
 public abstract TierRegistration saveTierRegistration(DSTRegistration
poDataDST,
                GMTInfoKeeper gMTInfoKeeper);

TierRegistration 1.1.1.3
Lines 13-15
 * @version $Id: TierRegistration.java,v 1.1.1.3 2014/08/23 23:17:57 sale_ahm
Exp $
 * * Code Refactored by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
 *   Abstract Function TierRegistration added as part of Polymorphism to be
implemented by Inherited classes.
```

Figure 67. Changesets and Diffs for Issue G- Part 5

Issue H:

```
package gipsy.GEE.multitier.GMT

GMTWrapper 1.1.1.1

Line 43
 * @version $Id: GMTWrapper.java,v 1.1.1.1 2014/08/14 02:11:02 mu_kumar Exp $

Line 106
private static final String MSG_PREFIX = "[" + Trace.getEnclosingClassName() +
"] ";

Lines 596-721
Method: allocateTier(String,TierIdentity,Configuration,DSTRegistration,int)

DSTRegistration oSysTAReg =  this.oInfoKeeper.getNodeSysDST(pstrNodeID);
                ITransportAgent oSysTA = oSysTAReg.getTA();

                if(oSysTA == null)
                {
                        oSysTA =
TAFactory.getInstance().createTA(oSysTAReg.getTAConfig());
                }

                TierAllocationRequest oRequest = new
TierAllocationRequest(pstrNodeID, poTierIdentity,
                                poTierConfig, piNumOfInstances);

                String strTierID =
poTierConfig.getProperty(IMultiTierWrapper.WRAPPER_TIER_ID);

                boolean bIsReallocation = false;

                if(strTierID != null)
                {
                        bIsReallocation = true;

                        if(Debug.isDebugOn())
                        {
                                Debug.debug(MSG_PREFIX + "This is a reallocation!");
                        }

                        if(piNumOfInstances != 1)
                        {
                                throw new MultiTierException("Reallocateion of multiple
tiers is not allowed!");
                        }
                }

                for(int i = 0; i<4; i++)
                {
                        try
                        {
                                oSysTA.setDemand(oRequest);
```

Figure 68. Changesets and Diffs for Issue H- Part 1

```
                        break;
                }
                catch (DMSException oException)
                {
                        if(i<3)
                        {
                                this.handleRegDSTCrash();
                        }
                        else
                        {
                                if(Debug.isDebugOn())
                                {
                                        oException.printStackTrace(System.err);
                                }
                                throw new MultiTierException(oException);
                        }
                }
        }
        if(Debug.isDebugOn())
        {
                Debug.debug(MSG_PREFIX + "The TierAllocationRequest " +
                                oRequest.getSignature().toString() + " was sent!");
        }
        TierAllocationResult oResult = null;
        for(int i = 0; i< 4 && oResult == null; i++)
        {
                try
                {
                        oResult =
(TierAllocationResult)oSysTA.getResult(oRequest.getSignature());
                }
                catch (DMSException oException)
                {
                        if(i<3)
                        {
                                this.handleRegDSTCrash();
                        }
                        else
                        {
                                if(Debug.isDebugOn())
                                {
                                        oException.printStackTrace(System.err);
                                }
                                throw new MultiTierException(oException);
                        }
                }
        }
        if(Debug.isDebugOn())
        {
                Debug.debug(MSG_PREFIX + "The TierAllocationResult " +
```

Figure 69. Changesets and Diffs for Issue H- Part 2

```
                                oResult.getSignature().toString() + " was
received!");
        }
        TierRegistration[] aoRegistrations = oResult.getRegistrations();
        if(aoRegistrations == null || aoRegistrations.length == 0)
        {
                if(oResult.getException() != null)
                {
                        throw oResult.getException();
                }
                else
                {
                        throw new MultiTierException("The allocation failed for
unknown reason!");
                }
        }
        for(int i = 0; i<aoRegistrations.length; i++)
        {
                TierRegistration oRegistration = aoRegistrations[i];
                if(poTierIdentity == TierIdentity.DST)
                {
                        DSTRegistration oDSTReg = (DSTRegistration)oRegistration;
                        if(bIsReallocation)
                        {
      oDSTReg.setMaxActiveConnection(poDataDSTReg.getMaxActiveConnection());
      oDSTReg.setActiveConnectionCount(poDataDSTReg.getActiveConnectionCount());
                        }
                        oDSTReg.setAllocationTime(System.currentTimeMillis());
                }
                this.oInfoKeeper.saveTierRegistration(oRegistration,
poDataDSTReg);
}
Lines 728-798
Method: deallocateTier(String, TierIdentity,String[])
DSTRegistration oSysTAReg =  this.oInfoKeeper.getNodeSysDST(pstrNodeID);
        ITransportAgent oSysTA = oSysTAReg.getTA();
        if(oSysTA == null)
        {
                oSysTA =
TAFactory.getInstance().createTA(oSysTAReg.getTAConfig());
        }
```

Figure 70. Changesets and Diffs for Issue H- Part 3

```
        TierDeallocationRequest oRequest = new
TierDeallocationRequest(pstrNodeID,
                        poTierIdentity, pastrTierIDs);
        for(int i = 0; i<4; i++)
        {
                try
                {
                        oSysTA.setDemand(oRequest);
                        break;
                }
                catch (DMSException oException)
                {
                        if(i<3)
                        {
                                this.handleRegDSTCrash();
                        }
                        else
                        {
                                if(Debug.isDebugOn())
                                {
                                        oException.printStackTrace(System.err);
                                }
                                throw new MultiTierException(oException);
                        }
                }
        }
        if(Debug.isDebugOn())
        {
                Debug.debug(MSG_PREFIX + "The TierAllocationRequest " +
                                oRequest.getSignature().toString() + " was sent!");
        }
        TierDeallocationResult oResult = null;
        for(int i = 0; i< 4 && oResult == null; i++)
        {
                try
                {
                        oResult =
(TierDeallocationResult)oSysTA.getResult(oRequest.getSignature());
                }
                catch (DMSException oException)
                {
                        if(i<3)
                        {
                                this.handleRegDSTCrash();
                        }
                        else
                        {
                                if(Debug.isDebugOn())
                                {
                                        oException.printStackTrace(System.err);
                                }
```

Figure 71. Changesets and Diffs for Issue H- Part 4

```
                                throw new MultiTierException(oException);
                        }
                }
        }
        if(Debug.isDebugOn())
        {
                Debug.debug(MSG_PREFIX + "The TierDeallocationResult " +
                                oResult.getSignature().toString() + " was
received!");
        }
Line 903
Method: handleNonRegDSTCrash(DSTRegistration)
this.allocateTier( poCrashedDSTReg.getNodeID(),
Line 801
Method: handleRegDSTCrash()
private void handleRegDSTCrash()
GMTWrapper 1.1.1.2
Line 43
 * @version $Id: GMTWrapper.java,v 1.1.1.2 2014/08/23 00:01:10 sale_ahm Exp $
Line 48
private GMTWrapperProduct gMTWrapperProduct = new GMTWrapperProduct();
Line 107
public static final String MSG_PREFIX = "[" + Trace.getEnclosingClassName() + "]
";
Lines 601-603
Method: allocateTier(String,TierIdentity,Configuration,DSTRegistration,int)
        gMTWrapperProduct
                .allocateTier(pstrNodeID, poTierIdentity, poTierConfig,
                        poDataDSTReg, piNumOfInstances, oInfoKeeper, this);
Lines 610-611
Method: deallocateTier(String, TierIdentity,String[])
gMTWrapperProduct.deallocateTier(pstrNodeID, poTierIdentity,
                        pastrTierIDs, oInfoKeeper, this);
Line 716
Method: handleNonRegDSTCrash(DSTRegistration)
        gMTWrapperProduct.allocateTier( poCrashedDSTReg.getNodeID(),
Line 604
Method: handleRegDSTCrash()
```

Figure 72. Changesets and Diffs for Issue H- Part 5

```
public void handleRegDSTCrash()
GMTWrapper 1.1.1.3
//Code changes added by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
      //Instance of the GMTWrapperProduct class added which was created to make
our God class.
      /*
       *Move Method to move methods over from GMTWrapper class to
GMTWrapperProduct class.
       *method: allocateTier: void
       *deallocateTier: void
       */
package gipsy.GEE.multitier.GMT
GMTWrapperProduct 1.1.2.1
New class added to the package.
Code is not provided as it occupies a lot of space and most of it is given
above.
GMTWrapperProduct 1.1.2.2
Lines 2-8
//Code changes added by Aditya Dewal, Ashish Arora, Kanwaldeep Singh, Lovepreet
Singh, Mukesh Kumar, Saleh Ahmed, Shivam Patel
//Instance of the GMTWrapperProduct class added which was created to make our
God class.
/*
 *Moved method to GMTWrapperProduct class from GMTWrapper class.
 *method: allocateTier: void
 *deallocateTier: void
 */
```

Figure 73. Changesets and Diffs for Issue H- Part 6

VIII. JUnit Test Cases

A. DMARF

As part of DMARF refactoring issue A, we change the class TextLoader and create the classes TextLoader_INgramModel, TextLoader_Bigram, TextLoader_Unigram and TextLoader_Trigram. As for these classes the test cases are already present in the TestLoaders class, so we do not create test cases for this refactoring.

For DMARF refactoring issue C, we change the file NeuralNetwork and Layer class. For these modules also the test cases are present as TestNN class so we do not create the JUnit test cases for these changes.

B. GIPSY

As part of GIPSY refactoring issue G, we change the classes GMTInfoKeeper, DGTRegistration, DWTRegistration and TierRegisteration. We do not prepare JUnit test cases for these changes for time constraint and complexity.

For GIPSY refactoring issue H, we change in the class GMTWrapper and create a class GMTWrapperProduct. For this change we create a JUnit test class as TestGMTWrapper where we are make an attempt to test the behaviour of all the functions in GMTWrapper and the functions which are pulled to GMTWrapperProduct class.

Source Code for JUnit Test Case: GIPSY Refactoring Issue H

```
package gipsy.tests.junit.GEE.multitier.GMT;

import static org.junit.Assert.*;
import junit.framework.Assert;
import gipsy.Configuration;
import gipsy.GEE.multitier.TierIdentity;
import gipsy.GEE.multitier.GMT.GMTWrapper;
import gipsy.GEE.multitier.GMT.demands.DSTRegistration;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestGMTWrapper {

	GMTWrapper oGMTWrapper1 = new GMTWrapper();
	GMTWrapper oGMTWrapper2 = new GMTWrapper();
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {}
	@AfterClass
	public static void tearDownAfterClass() throws Exception {}
	@Before
	public void setUp() throws Exception {}
	@After
	public void tearDown() throws Exception {}
	@Test
	public final void testRun() {
		try{
			oGMTWrapper1.run();
			oGMTWrapper2.run();
		}catch(Exception e){
			e.printStackTrace();
		}
		Assert.assertTrue(oGMTWrapper1.equals(oGMTWrapper2));
		Assert.assertFalse(oGMTWrapper1.equals(oGMTWrapper2));
	}

	@Test
	public final void testStartTier() {
		try{
			oGMTWrapper1.startTier();
			oGMTWrapper2.startTier();
		}catch(Exception e){
			e.printStackTrace();
		}
		Assert.assertTrue(oGMTWrapper1.equals(oGMTWrapper2));
		Assert.assertFalse(oGMTWrapper1.equals(oGMTWrapper2));
	}

	@Test
	public final void testAllocateTier() {
		try{
			String pstrNodeID = new String(); // Dummy String Value
			TierIdentity poTierIdentity = null; //Dummy TierIdentity Value
```

Figure 74. Source code for JUNit Test-Part 1

```
			Configuration poTierConfig = null;
			DSTRegistration poDataDSTReg = null;
			int piNumOfInstances = 0; //Dummy piNumOfInstances value
			oGMTWrapper1.allocateTier(pstrNodeID, poTierIdentity,
poTierConfig, poDataDSTReg, piNumOfInstances);
			oGMTWrapper2.allocateTier(pstrNodeID, poTierIdentity,
poTierConfig, poDataDSTReg, piNumOfInstances);
		}catch(Exception e){
			e.printStackTrace();
		}
		Assert.assertTrue(oGMTWrapper1.equals(oGMTWrapper2));
		Assert.assertFalse(oGMTWrapper1.equals(oGMTWrapper2));
	}

	@Test
	public final void testDeallocateTier() {
		try{
			String pstrNodeID = new String(); // Dummy String Value
			TierIdentity poTierIdentity = null; //Dummy TierIdentity Value
			String []pastrTierIDs =null; //Dummy piNumOfInstances value
			oGMTWrapper1.deallocateTier(pstrNodeID, poTierIdentity,
pastrTierIDs);
			oGMTWrapper2.deallocateTier(pstrNodeID, poTierIdentity,
pastrTierIDs);
		}catch(Exception e){
			e.printStackTrace();
		}
		Assert.assertTrue(oGMTWrapper1.equals(oGMTWrapper2));
		Assert.assertFalse(oGMTWrapper1.equals(oGMTWrapper2));
	}

	@Test
	public final void testHandleRegDSTCrash() {
		try{
			oGMTWrapper1.handleRegDSTCrash();
			oGMTWrapper2.handleRegDSTCrash();
		}catch(Exception e){
			e.printStackTrace();
		}
		Assert.assertTrue(oGMTWrapper1.equals(oGMTWrapper2));
		Assert.assertFalse(oGMTWrapper1.equals(oGMTWrapper2));
	}

	@Test
	public final void testStopTier() {
		try{
			oGMTWrapper1.stopTier();
			oGMTWrapper2.stopTier();
		}catch(Exception e){
			e.printStackTrace();
		}
		Assert.assertTrue(oGMTWrapper1.equals(oGMTWrapper2));
		Assert.assertFalse(oGMTWrapper1.equals(oGMTWrapper2));
	}
}
```

Figure 75. Source code for JUNit Test-Part 2

IX. Conclusion

The classical MARF is protracted to allow the stages of the pipeline to run as distributed nodes as well as their front-ends. The basic stages and the frontends are designed to provision, but implemented without backup recovery or hot-swappable capabilities. They only support communication over Java RMI, CORBA, and XML-RPC WebServices. GIPSY is instigated to investigate properties of the Lucid family of intensional programming languages and beyond. It is a distributed system, designed as a modular pool of frameworks where components linked to the development and execution of Lucid programs, are decoupled to permit tranquil extension, addition, and replacement of the components. GIPSY has a collection of compilers under GIPC and the conforming run-time environment under GEE among other properties which communicate through GEER.

On analysis of the conceptual domain model versus the actual domain model of GIPSY and DMARF, we find out that no matter how exact knowledge of the domain the analysts may portray the actual domain model/architecture oath to have some difference. So, it is always better to analyze using reverse engineering tools for classes and dependencies. Further, while doing the fusion of the multi-tier domains of GIPSY and pipeline domains of DMARF, DoGRTA, we find that architecture of DMARF can be turned into multi-tier giving it a better cohesion, coupling and separation of concerns.

While Identifying components to be refactored for removal of code smells we emphasise on the modules which are not having a lot of dependencies i.e. the modules are having less association in comparison to other modules in the projects. This is done such that the impact of the refactoring is minimal and the behavioural changes can be easily identified. After following this approach we find that, in DMARF some of the classes of modules, NeuralNetwork and TextLoader can be refactored with minimal changes in other components. Similarly, we identify modules GMTWrapper for refactoring in GIPSY.

During the code analysis methodology, to find out the code smells and to refactor the code such that the code smells could be eliminated, we found out that doing refactoring in itself when guided by tools like JDeodrant, is not as intensive task as generating the test cases for the refactoring. Therefore verifying that the behaviour of system is not changed became an intensive and manual task.

In case of Test Driven Development/Refactoring as the test cases are developed prior to the changes so, the behaviour of the system can be easily verified after the changes. As test cases for the DMARF project are already present in the system the verification of the behaviour after changes is not a cumbersome task but it is not the same in the case of GIPSY.

REFERENCES

[1] Yi Ji, Serguei A. Mokhov, and Joey Paquet. Unifying and refactoring DMF to support concurrent Jini and JMS DMS in GIPSY. In Bipin C. Desai, Sudhir P. Mudur, and Emil I. Vassev, editors, *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering* (C3S2E'12), pages 36–44, New York, NY, USA, June 2010–2013. ACM. ISBN 978-1-4503-1084-0. doi: 10.1145/2347583.2347588. Online e-print http://arxiv.org/abs/1012.2860.

[2] Serguei A. Mokhov. On design and implementation of distributed modular audio recognition frame- work: Requirements and specification design document. [online], August 2006. Project report, http://arxiv.org/abs/0905.2459, last viewed April 2012.

[3] Serguei A. Mokhov. Towards security hardening of scientific distributed demand-driven and pipelined computing systems.In *Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC'08)*, pages 375–382. IEEE Computer Society, July 2008. ISBN 978-0-7695-3472- 5. doi: 10.1109/ISPDC.2008.52.

[4] Serguei A. Mokhov and Rajagopalan Jayakumar. Distributed Modular Audio Recognition Frame- work (DMARF) and its applications over web services. In Tarek Sobh, Khaled Elleithy, and Ausif Mahmood, editors, *Proceedings of TeNe'08*, pages 417–422, University of Bridgeport, CT, USA, De- cember 2008. Springer. ISBN 978-90-481-3661-2. doi: 10.1007/978-90-481-3662-9 72. Printed in January 2010.

[5] Serguei A. Mokhov and Emil Vassev. Autonomic specification of self-protection for Distributed MARF with ASSL. In *Proceedings of C3S2E'09*, pages 175–183, New York, NY, USA, May 2009. ACM. ISBN 978-1-60558-401-0. doi: 10.1145/1557626.1557655

[6] Serguei A. Mokhov and Emil Vassev. Self-forensics through case studies of small to medium software systems. In *Proceedings of IMF'09*, pages 128–141. IEEE Computer Society, September 2009. ISBN 978-0-7695-3807-5. doi: 10.1109/IMF.2009.19.

[7] Serguei A. Mokhov, Lee Wei Huynh, and Jian Li. Managing distributed MARF with SNMP. Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, April 2007. Project report. Hosted at http://marf.sf.net and http://arxiv.org/abs/0906.0065, last viewed February 2011.

[8] Serguei A. Mokhov, Emil Vassev, Joey Paquet, and Mourad Debbabi. Towards a self-forensics property in the ASSL toolset. In *Proceedings of the Third C* Conference on Computer Science and Software Engineering (C3S2E'10)*, pages 108–113, New York, NY, USA, May 2010. ACM. ISBN 978-1-60558-901-5. doi: 10.1145/1822327.1822342.

[9] Joey Paquet. Distributed eductive execution of hybrid intensional programs. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMP- SAC'09)*, pages 218–224. IEEE Computer Society, July 2009. ISBN 978-0-7695-3726-9. doi: 10.1109/COMPSAC.2009.137.

[10] Sleiman Rabah, Serguei A. Mokhov, and Joey Paquet. An interactive graph-based automation assistant: A case study to manage the GIPSY's distributed multi-tier run-time system. In Ching Y. Suen, Amir Aghdam, Minyi Guo, Jiman Hong, and Esmaeil Nadimi, editors, *Proceedings of the ACM Research in Adaptive and Convergent Systems (RACS 2013)*, pages 387–394, New York, NY, USA, October 2011–2013. ACM. ISBN 978-1-4503-2348-2. doi: 10.1145/2513228.2513286. Preprint: http://arxiv.org/abs/1212.4123.

[11] Emil Vassev and Serguei A. Mokhov. Self-optimization property in autonomic specification of Distributed MARF with ASSL. In Boris Shishkov, Jose Cordeiro, and Alpesh Ranchordas, editors, *Proceedings of ICSOFT'09*, volume 1, pages 331–335, Sofia, Bulgaria, July 2009. INSTICC Press. ISBN 978-989-674-009-2. doi: 10.5220/0002257303310335.

[12] Emil Vassev and Serguei A. Mokhov. Towards autonomic specification of Distributed MARF with ASSL: Self-healing. In *Proceedings of SERA 2010 (selected papers)*, volume 296 of SCI, pages 1–15. Springer, 2010. ISBN 978-3-642-13272-8. doi: 10.1007/978-3-642-13273-5 1.

[13] Emil Vassev and Joey Paquet. Towards autonomic GIPSY. In *Proceedings of the Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASE 2008)*, pages 25–34. IEEE Computer Society, April 2008. ISBN 978-0-7695-3140-3. doi: 10.1109/EASe.2008.9.

[14] Emil Iordanov Vassev. General architecture for demand migration in the GIPSY demand-driven execution engine. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, June 2005. ISBN 0494102969.

[15] Intooitus Developers.,inCode. Intooitus, 2008-2014. https://www.intooitus.com/products/incode.

[16] Eclipse Technology., CodePro Analytix. Google, Inc., 2001-2010. https://developers.google.com/java-dev-tools/codepro/doc.

[17] Frank Sauer.,Eclipse Metrics plugin. sourceforge.net, 2014. http://metrics.sourceforge.net.

[18] Ontario Provincial Police.,http://www.opp.ca.

[19] Serguei A. Mokhov. Intensional Cyberforensics. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, September 2013. Online at http://arxiv.org/abs/1312.0466.

[20] Serguei A. Mokhov. Towards Improving Validation, Verification, Crash Investigations, and Event Reconstruction of Flight-Critical Systems with Self-Forensics. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, June 2009. Online at http://arxiv.org/abs/0906.1845

[21] Linus Tolke., ArgoUML v0.34. Eclipse, 1996-2009. http://argouml-users.net/index.php?title=Main_Page

[22] Transportation Safety Board of Canada, 2014., http://www.tsb.gc.ca/eng/index.asp

[23] The MARF Research and Development Group., MARF, The Modular Audio Recognition Framework, and its Applications. Concordia University, 2002 – 2008. http://marf.sourceforge.net/

[24] The GIPSY Research and Development Group., GIPSY: A General Intensional Programming System. Concordia University, 2000-2011. http://newton.cs.concordia.ca/~gipsy/

[25] Pascal Brachet.,Texmaker 4.2 (LaTeX). GNU GPL License, 2003-2014. http://www.xm1math.net/texmaker/

[26] Alexander Shvets.,CC License, 2006-2014.http://sourcemaking.com/refactoring

[27] JDeodorant Team.,http://www.jdeodorant.com/~JDeodorant

[28] ObjectAid UML Explorer for Eclipse.,http://www.objectaid.com

[29] Design Patterns.,http://www.oodesign.com

[30] Software design pattern.,http://en.wikipedia.org/wiki/Software_design_pattern

[31] Eclipse Luna.,The Eclipse Foundation, 2014.https://www.eclipse.org

[32] Lubomir Elko.,Design Pattern Recognizer, FIIT SUT in Bratislava, 2011. http://lubes.yweb.sk/projects/dprecognizer/

X. APPENDIX

A. Code Analysis Snapshots

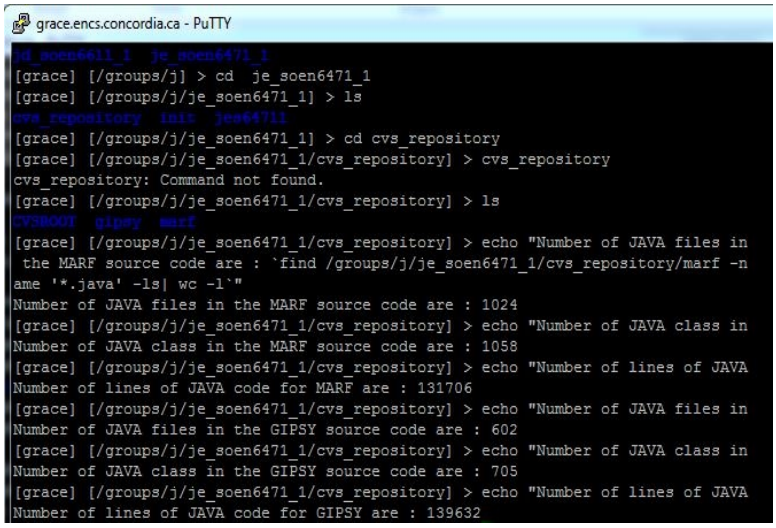

Figure 76. Code analysis using Linux

GIPSY at 7/22/14 5:07 PM

Metric
Abstractness
Average Block Depth
Average Cyclomatic Complexity
Average Lines Of Code Per Method
Average Number of Constructors Per Type
Average Number of Fields Per Type
Average Number of Methods Per Type
Average Number of Parameters
Comments Ratio
Efferent Couplings
Lines of Code
Number of Characters
Number of Comments
Number of Constructors
Number of Fields
Number of Lines
Number of Methods
Number of Packages
Number of Semicolons
Number of Types
Weighted Methods

Figure 77. Measures of GIPSY using CodePro Analytix

Name	Value
interface	76
public	73
protected	0
package	3
private	0
class	626
public	540
protected	0
package	65
private	21

Figure 78. Measures of GIPSY (Number of Classes) using CodePro Analytix

marf at 7/23/14 2:08 PM

Metric	Value
Abstractness	12%
Average Block Depth	1.01
Average Cyclomatic Complexity	1.75
Average Lines Of Code Per Method	8.49
Average Number of Constructors Per Type	1.18
Average Number of Fields Per Type	1.85
Average Number of Methods Per Type	5.95
Average Number of Parameters	1.12
Comments Ratio	14.4%
Efferent Couplings	842
Lines of Code	77,297
Number of Characters	4,878,311
Number of Comments	11,164
Number of Constructors	1,249
Number of Fields	4,237
Number of Lines	131,772
Number of Methods	6,305
Number of Packages	125
Number of Semicolons	36,470
Number of Types	1,058
Weighted Methods	14,083

Figure 79. Measures of DMARF using CodePro Analytix

Percentages by kind | Breakdown by kind | Percentag

Name	Value
interface	75
public	75
protected	0
package	0
private	0
class	983
public	970
protected	2
package	6
private	5
	26,758

Figure 80. Measures of DMARF(Number of Classes) using CodePro Analytix

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum
Number of Overridden Methods (avg/max per type)	439	0.757	1.761	21	/GIPSY/src/gipsy/GIPC/intensional/SIPL/ForensicLuci...
Number of Attributes (avg/max per type)	1851	3.191	5.104	33	/GIPSY/src/gipsy/RIPE/editors/RunTimeGraphEditor/...
Number of Children (avg/max per type)	236	0.407	1.812	21	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/ast/ex...
Number of Classes (avg/max per packageFragment)	580	5.472	5.572	36	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/ast/ex...
Method Lines of Code (avg/max per method)	75262	12.44	40.732	933	/GIPSY/src/gipsy/GIPC/SemanticAnalyzer.java
Number of Methods (avg/max per type)	5746	9.907	33.947	555	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/JavaP...
Nested Block Depth (avg/max per method)		1.71	1.313	19	/GIPSY/src/gipsy/GIPC/intensional/SIPL/Lucx/LucxPa...
Depth of Inheritance Tree (avg/max per type)		2.469	1.683	8	/GIPSY/src/gipsy/GEE/multitier/GMT/demands/DWT...
Number of Packages	106				
Afferent Coupling (avg/max per packageFragment)		10.943	19.952	90	/GIPSY/src/gipsy/GEE/IDP/demands
Number of Interfaces (avg/max per packageFragment)	76	0.717	1.242	6	/GIPSY/src/gipsy/lang/context
McCabe Cyclomatic Complexity (avg/max per method)		4.055	13.426	300	/GIPSY/src/gipsy/GIPC/intensional/GIPL/GIPLParserT...
Total Lines of Code	104073				
Instability (avg/max per packageFragment)		0.598	0.36	1	/GIPSY/src/gipsy/GEE/IDP/DemandGenerator/rmi
Number of Parameters (avg/max per method)		0.794	1.099	11	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/ast/b...
Lack of Cohesion of Methods (avg/max per type)		0.243	0.361	1.625	/GIPSY/src/gipsy/GIPC/util/Token.java
Efferent Coupling (avg/max per packageFragment)		4.377	4.721	34	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/ast/ex...
Number of Static Methods (avg/max per type)	302	0.521	1.561	14	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/ast/b...
Normalized Distance (avg/max per packageFragment)		∅	∅	1	/GIPSY/src/gipsy/RIPE/RuntimeSystem
Abstractness (avg/max per packageFragment)		∅	∅	0.75	/GIPSY/src/gipsy/GEE/IDP/DemandDispatcher
Specialization Index (avg/max per type)		0.542	0.953	5.04	/GIPSY/src/gipsy/tests/GEE/IDP/demands/DemandTe...
Weighted methods per Class (avg/max per type)	24533	42.298	185.189	2856	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/JavaP...
Number of Static Attributes (avg/max per type)	836	1.441	4.938	76	/GIPSY/src/gipsy/GIPC/intensional/SIPL/JOOIP/JavaP...

Figure 81. Measures of GIPSY using Metrics

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per type)	703	0.718	1.213	10	/marf/src/marf/math/ComplexMatrix.java	
Number of Attributes (avg/max per type)	1675	1.711	5.456	101	/marf/src/marf/net/server/ws/MARF/IMARFServerWS...	
Number of Children (avg/max per type)	203	0.207	0.998	13	/marf/src/marf/Storage/StorageManager.java	
Number of Classes (avg/max per packageFragment)	979	8.513	24.43	228	/marf/src/marf/net/server/ws/MARF	
Method Lines of Code (avg/max per method)	44441	6.28	18.004	708	/marf/src/marf/net/server/ws/MARF/IMARFWS_Servi...	getRegistry
Number of Methods (avg/max per type)	6109	6.24	8.348	102	/marf/src/marf/net/server/ws/MARF/IMARFServerWS...	
Nested Block Depth (avg/max per method)		1.322	0.804	10	/marf/src/marf/nlp/Parsing/ProbabilisticParser.java	parse
Depth of Inheritance Tree (avg/max per type)		2.11	1.192	6	/marf/src/marf/Preprocessing/CFEFilters/HighPassFil...	
Number of Packages	115					
Afferent Coupling (avg/max per packageFragment)		9.217	25.349	187	/marf/src/marf/util	
Number of Interfaces (avg/max per packageFragment)	72	0.626	0.849	4	/marf/src/marf/Storage	
McCabe Cyclomatic Complexity (avg/max per method)		1.82	2.641	60	/marf/src/marf/net/server/corba/MARF/IMARFServer...	_invoke
Total Lines of Code	77297					
Instability (avg/max per packageFragment)		0.488	0.31	1	/marf/src	
Number of Parameters (avg/max per method)		1.187	1.268	15	/marf/src/marf/Configuration.java	Configuration
Lack of Cohesion of Methods (avg/max per type)		0.175	0.296	1.333	/marf/src/marf/nlp/Parsing/VarSymTabEntry.java	
Efferent Coupling (avg/max per packageFragment)		6.696	17.345	155	/marf/src/marf/net/server/ws/MARF	
Number of Static Methods (avg/max per type)	968	0.989	9.278	279	/marf/src/marf/util/Arrays.java	
Normalized Distance (avg/max per packageFragment)		0.4	0.29	1	/marf/src/marf/net/assl/generatedbyassl/as/dmarf	
Abstractness (avg/max per packageFragment)		0.182	0.267	1	/marf/src/marf/net/client/corba	
Specialization Index (avg/max per type)		0.38	0.598	3.2	/marf/src/marf/Classification/Markov/Markov.java	
Weighted methods per Class (avg/max per type)	12880	13.156	28.198	426	/marf/src/marf/net/server/ws/MARF/IMARFServerWS...	
Number of Static Attributes (avg/max per type)	2511	2.565	11.959	223	/marf/src/marf/net/server/ws/MARF/IMARFServerWS...	

Figure 82. Measures of DMARF using Metrics

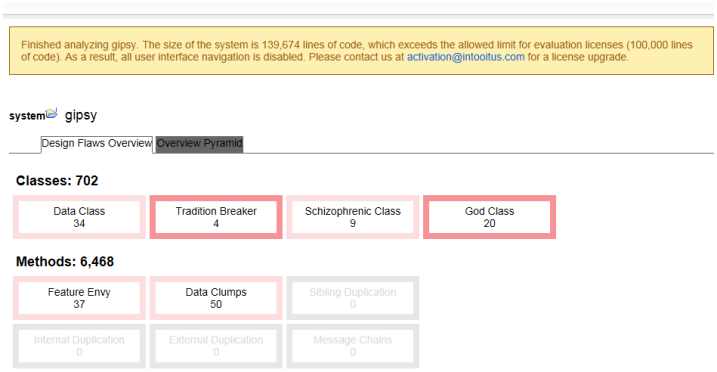

Figure 83. Measures of GIPSY using inCode

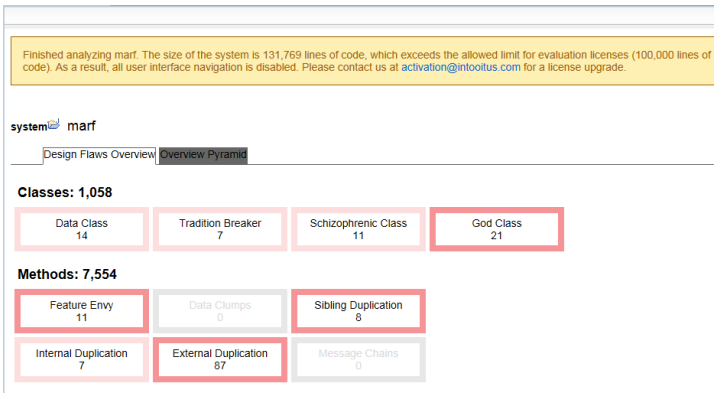

Figure 84. Measures of DMARF using inCode

Table V
DESIGN PATTERN DISTRIBUTION

Serial No.	Name	Design Pattern
1	Aditya Dewal	Factory (Creation GoF Pattern)
2	Ashish Arora	State (Behavioral GoF pattern)
3	Kanwaldeep Singh	Stratergy (Behavioral GoF pattern)
4	Lovepreet Singh	Facade (Structural GoF pattern)
5	Mukesh Kumar	Observer (Behavioral GoF pattern)
6	Saleh Ahmed	Composite (Structural GoF pattern)
7	Shivam Patel	Singleton (Creational GoF pattern)

Table VI
ARTICLE DISTRIBUTION: DMARF

Serial No.	Name	Design Pattern
1	Aditya Dewal	On design and implementation of distributed modular audio recognition framework: Requirements and specification design document[2]
2	Ashish Arora	Towards autonomic specification of Distributed MARF with ASSL: Self-healing[12]
3	Kanwaldeep Singh	Self-optimization property in autonomic specification of Distributed MARF with ASSL[11]
4	Lovepreet Singh	Autonomic specification of self-protection for Distributed MARF with ASSL[5]
5	Mukesh Kumar	Managing distributed MARF with SNMP[7]
6	Saleh Ahmed	Distributed Modular Audio Recognition Framework (DMARF) and its applications over web services[4]
7	Shivam Patel	Towards security hardening of scientific distributed demand-driven and pipelined computing systems[3]

Table VII
ARTICLE DISTRIBUTION: GIPSY

Serial No.	Name	Design Pattern
1	Aditya Dewal	Self-forensics through case studies of small to medium software systems[6]
2	Ashish Arora	General architecture for demand migration in the GIPSY demand-driven execution engine[14]
3	Kanwaldeep Singh	Towards a Self-forensics property in the ASSL toolset[8]
4	Lovepreet Singh	Unifying and refactoring DMF to support concurrent Jini and JMS DMS in GIPSY[1]
5	Mukesh Kumar	Towards Autonomic GIPSY[13
6	Saleh Ahmed	An interactive graph-based automation assistant: A case study to manage the GIPSY's distributed multi-tier run-time system[10]
7	Shivam Patel	Distributed eductive execution of hybrid intentional programs[9]