

Multilingual Source Code Analysis: State of the Art and Challenges

Zaigham Mushtaq¹, Ghulam Rasool²

Department of Computer Science
COMSATS Institute of Information Technology
Lahore, Pakistan
zmqazi@gmail.com¹, grasool@ciitlahore.edu.pk²

Abstract— Source code analysis and manipulation for multilingual types of software applications is an important research topic and its importance is continuously increasing. As utilization of multilingual source code applications is increasing, the demand of multilingual source code analysis is escalating. We discuss the widespread applications of multiple technologies and languages used for development of software systems and difficulties for analyzing the source code of these applications in this paper. We highlight the key challenges and open research problems in the field of multilingual software applications that require the attention of research community.

Keywords— *Source code analysis; Cross-language; Analysis methods; Reverse Engineering; Source code parsing*

I. INTRODUCTION

Source code analysis is an important artifact for software applications and its importance will increase due to the development of new heterogeneous nature of software applications which have impact on the biological, socioeconomic, legal and governmental processes of interconnected world. It is estimated that size of software in 2025 will be more than 1 trillion lines of code [4] that reflects the importance of source code analysis and manipulation in future [16]. There is a continuous trend to translate high level and abstract descriptions towards executability that will ultimately increase the size of source code. Source code analysis yields benefits to different disciplines such as reverse engineering, reengineering, maintenance, comprehension, refactoring, debugging, re-documentation etc. 29 applications of source code analysis for different domains of software engineering are highlighted in [4]. Static, dynamic, and historic source code analysis methods are used in literature to analyze the source code of software applications for different objectives.

Static source code analysis methods focus on extracting structural relationships such as inheritance, association, friend relationships, interface hierarchies, modifiers of classes and methods, method parameters, method return types, attributes, data types, etc. These methods are still limited to analysis of source code for single languages and they are not yet capable to extracting complete dynamic aspects from the source code of heterogeneous applications. The dynamic source code analysis methods are used to extract runtime information from the

source code. The major difficulty in dynamic analysis is that there can be various possible implementations for the same expected behavior. The dynamic analysis methods focus on capturing an execution history and control flow traces. Examples of dynamic information are execution traces, log files and error messages etc. It is often desirable to have both static and dynamic analysis to get a holistic picture of examined applications. During the last 10 years, software analysis moved from a single dimension (static analysis) to two dimensions (static and dynamic analysis) and finally today's opportunity is to add a third dimension consisting in the historical analysis of data extracted from software repositories [8].

The trend for developing different software applications based on source code of more than one programming languages is continuously increasing. Embedded systems, web applications, mobile applications and J2EE patterns are developed using multiple technologies and languages. Different editors and frameworks such as EditRocket, NetBeans, Visual LANSa, .Net and play framework support multiple languages for development of software applications. For example, Netbeans is an open source that provides smarter and faster IDE to code applications using different languages such as Java, PHP, JSP, C/C++, Groovy and Apache. It is predicted by different authors [17, 34, 39] that multilingual source code analysis will get more attention of research community in the future due to complex heterogeneous structures of modern applications. The recovery of artifacts from such applications is challenging as most of existing source code analysis methods focus on extracting information from applications which are developed using single languages.

Source code analysis is a prerequisite for a number of disciplines such as re-engineering, reverse engineering, maintenance, refactoring, program comprehension etc. Existing source code analysis methods often have geared toward analysis of applications that are developed based on a single language. However, complex embedded systems, enterprise applications (J2EE environment) and web applications have heterogeneous structures. For example, the Philips MRI scanner uses many languages such as C, C++/STL, C#, Visual Basic and Perl [1]. Similarly, most web applications integrate different technologies such as scripting languages, middleware, web services, data warehouses and databases. The analysis of multilingual source code applications cannot yet get attention

due to the monolithic nature of software applications in past. The importance of multilingual source code analysis is realized by different researchers [4, 14, 16, 17, 34, 39]. There are various technologies that are found in multiple languages in the form of heterogeneous applications (e.g., Java, XML, SQL etc.) [17, 33]. We discuss the state of the art techniques and tools used for multilingual applications with their strengths and limitations in Section II. The challenges for the analyzing source code of multiple languages are discussed in Section III. The conclusion of the whole study is presented in Section IV.

II. STATE OF THE ART TECHNIQUES AND TOOLS

Many techniques and tools have been presented that support program understanding and analysis of multi-language software applications. To the best of our knowledge, there is no study presented in the literature that discussed multilingual source code analysis techniques and tools. We discuss techniques and tools that are capable of dealing with the complexity and heterogeneity of modern software systems below:

A. GUPRO (*Generic Understanding of Programs*)

GUPRO [13, 20] is “an integrated workbench that supports program understanding of multiple source code systems” [20]. The approach described in GUPRO implements graph-based conceptual modeling [20]. GUPRO is based on an extended entity-relationship dialect (EER) and the GRAL that can be used to perform architectural constraint validation and is focused on the analysis of multi-language systems [13]. Although GUPRO is able to handle the multi-language applications but its application is limited to analyze the source code of existing languages.

B. Multi-Language Metrics Tool (MMT)

A multi-language metrics extraction tool based on Microsoft Intermediate Language is presented by Linos et al. [21] to analyze multilingual applications. Authors proposed a set of multi-language software metrics which are based on the analysis of the Microsoft Intermediate Language (MSIL) [21]. They showed that the analysis of the MSIL is effective for .NET platform. The approach has advantages on other approaches based on an intermediate representation but contains drawbacks as it is not extensible and it only supports languages of .NET environment.

C. MoDISCO

MoDISCO [27] is an eclipse plug-in that supports the modernization of legacy multiple source code applications. The process in MoDISCO comprises an initial understanding phase based on a model of the legacy system under analysis. MoDISCO provides the possibility to define new models. Meta-modeling of MoDISCO is based on two standards: 1st one is Knowledge Discovery Meta model (KDM) and 2nd is Software Metrics Meta- model (SMM).

D. Knowledge Discovery Meta model (KDM)

KDM [27, 28] provides a complete set of meta-models that describe existing software systems that can be used in software

quality assurance and modernization. The KDM model provides twelve packages that focus on different aspects of modern software systems [27]. Although KDM is a candidate for multilingual analysis but it need a simpler model which could be easily extendable to support multilingual source code analysis [28].

E. Language Independent Meta-Model (X-Develop)

Strein et al. [33] proposed an approach for multi-language source code analysis that is based on a language independent meta-model. They design abstract representations and represent them in a language or paradigm. Authors proposed generic language descriptions to support multi-language refactoring with the help of a tool called X-Develop. The X-Develop model supports each language representation through front-end that is then transformed into a multi-language model. The drawback of this approach is that abstract representation of each language would restrict possible software analysis which could be done on the source code.

F. DATES(*Design Analysis Tool for Enterprise Applications*)

Marinescu et al. [24] recognized the need to model heterogeneous multiple source code applications as a whole though their meta-model, but this approach focused on the analysis of relational databases [25, 26]. Authors proposed a meta-model for enterprise applications which focuses only on object-oriented software systems and relational databases [24]. The approach is able to identify the relational discrepancies between database schemas and source code in enterprise applications [26].

G. Moose Platform

Moose platform[28] is a self-described core model like KDM as an extended model that is based on essential Meta Object Facility (EMOF) [28]. There is a family of meta-models that are used to describe different aspects of a modern software application in the architecture of Moose. These models enable analysis and contain a rich collection of API's that can be used for querying and navigation. The process in Moose is focused on the analysis and understanding of multilingual systems. However, more detailed elements are needed that could effectively describe the real architecture of multiple language/heterogeneous systems.

H. Island Grammars

Island Grammars [34] is proposed for source code analysis and reverse engineering of multilingual applications. Island Grammar divides source code of examined applications into two categories: Islands, that provide detailed productions for the constructs of interest and Water that consists of the remaining insignificant parts [15]. The Grammar can be used when the parse tree is not complete, but it produces false negatives when used Grammar is not enough liberal. It is also labor intensive task to define a grammar for every new analysis task.

I. WARE(*Web Application Reverse Engineering*)

WARE [15] concept is based on GMT (Goals/Models/Tools)[3] paradigm to specify a reverse

engineering process for comprehension and recovery of information from web applications. Authors apply four-step process including static analysis, dynamic analysis, clustering and abstraction to recover class, sequence and use case diagrams from web applications. The approach seems mature and is validated through a case study, but it is only limited to source code analysis of web applications.

J. REAJAX

Authors present a reverse engineering approach supplemented with a tool named REAJEX to extract GUI models from Ajax based web applications [23]. They use dynamic analysis and execution traces to generate finite state machine models for GUI applications that are helpful for the comprehension of applications. The approach is validated with a case study on five real Ajax applications. However, the presented approach is limited to only Ajax based web applications and its generalization for other types of multilingual source code applications is questionable.

K. Analizo

Analizo [35] toolkit is composed of layered architecture that is used to analyze and visualize source code of C, C++ and Java languages with the support of Doxygen. The toolkit is available freely for experimentation. It supports extraction of source code metrics, generation of dependency graph and evolution analysis for applications that are developed using these three languages.

L. RASCAL

RASCAL [40] is among domain specific languages (DSLs), and it is used for analysis and manipulation of source code. It includes domain specific languages ASF, SDF, RScript and is easily extensible with Java code. RASCAL offers high class incorporation of source code analysis based on technical, syntactic, conceptual and semantic limits. It also offers a simple customizable and extendable Eclipse based environment(IDE).

TABLE I. COMPARISON OF MULTILINGUAL SOURCE CODE ANALYSIS TECHNIQUES/TOOLS

S #	Technique/Tool	Model/Representation	Analysis Mechanism	Source Code Languages	Experimental Case Studies
1	GUPRO (Generic Understanding of Programs). [13, 20].	Graph Repository	Source Code parsing based on graph-queries.	COBOL, CSP, MVS/JCL, PSB, SQL, C, Ada	GEOS, XFIG, COBOL examples, etc.
2	MMT [21]	MSIL(Microsoft Intermediate Language Level)	Parsing MSIL	.Net based languages	NHibernate, MMT(self), Timecard CS Client
3	MoDISCO an Eclipse Plug [27]	Navigational models MVC Web Frameworks	Static Analysis	HTML, JSP XML, Java	http://www.unex.es/eweb/migra/ria/cs/agenda
4	KDM [27, 28]	Meta-model(MOF)	-	OOPs, XML	Not Mentioned
5	Language Independent Meta-Model (X-Develop) [33]	Language specific Common Models	Conservative static low-level analyses)	C#, J#, VB	ITextDotNet
6	DATES [24]	DATES(a meta model)	Third party API	Java, SQL	KITTA, TRS, SALARY
7	MOOSE Platform [36]	FAMIX model	Type related Analysis	Java, Smalltalk	Small examples of Java and Smalltalk
8	Island Grammar [34]	CFG Rascal	Lexical Analysis	VisualBasic HTML, JavaScript	McCabe-complexity
9	WARE[15]	Re MDWE Meta-Model	Static and Dynamic	HTML, JavaScript, XML	General Examples
10	REAJAX[23]	DOM GUI-based State models	Dynamic and Trace Analysis	HTML, CSS Javascript, PHP XUL,XML	[http://pafm.sourceforge.net], [http://tudu.sourceforge.net]
11	Analizo toolkit [35]	Layered Style	Doxygen based on Doxygen	C, C++, Java	VLC project, Analizo(itsself) etc.
12	RASCAL [40]	Abstract Syntax	Lexical	ASF, SDF, RScript, Java	Eclipse IDE

III. CHALLENGES OF SOURCE CODE ANALYSIS

The crux of discussion in this section is that none of the previously mentioned techniques/tools provide a generic and extensible solution to support the analysis of multiple source code elements at multiple levels of abstraction in single application. The representation of multilingual applications is not complete and needs to be extended. The existing techniques/tools are difficult to extend and they are complex to understand. The source code analysis of complex, cross language applications is still a challenge for reverse engineering community [7, 17, 33]. It is however argued that a simpler model which is easier to extend is still required that support multilingual source code analysis.

Researchers in Bell Labs are currently conducting research in automated source code analysis and according to them, key challenges remain in areas such as analysis, accuracy, performance, scalability, flexibility, and applicability in real-world development environments. The key source code analysis challenge is perceived in cross language or multi-language applications. The challenges are aggregated in the case of large and complex systems that contain different languages and domains. The basic difficulty for source code parsing tools is to exactly recognize where the code of one language ends and other language begins. Tokenization, whitespaces and comments are difficult to parse because they have different treatments for each language [34]. It is also

realized that applications developed in multilingual platforms hide parts of source code and execution behavior which impede task of source code analysis tools. Other difficulties such as non-executable source code, hidden dependencies, source code having errors, missing middleware components further complicate the task of source code analysis tools. New source code analysis tools must be able to cope with the diversity of languages and technologies used to develop a single software system [7]. For example, web applications are often composed of HTML fragments, server side scripting, client side scripting and data- base queries written in SQL [10, 33]. New forms of programs will represent the source for future applications of reverse engineering [22].

The requirement of multilingual source code analysis in modern development environments such as (J2EE environment) is increasing as most systems and their components being developed are composed of many languages [33]. The J2EE platform is a multilingual platform that includes J2EE components such as Enterprise Java Beans (EJB), Java Server Pages (JSP), and Servlets. In order to analyze source code of J2EE platform, there must be a multilingual source code analysis support, that have the definition of J2EE patterns in the form of formal specification, and provision to detect J2EE patterns within the source code. To the best of our knowledge, source code parsers are not yet available which can analyze and detect J2EE patterns from different applications. There is a need to develop source code analysis approaches and tools for new software architectures containing the characteristics i.e, extremely dynamic, highly distributed, self-configurable and heterogeneous [7].

Embedded systems have widespread applications in different areas such as consumer electronics, household appliances, automobiles, medical devices, avionics systems, communications etc. A glance through the literature reveals that embedded systems are developed using multiple languages such as Assembly, C/C++, PLEX, COBOL, Erlang, mbeddr [37] etc. Source code analysis methods and tools have idiosyncratic challenges due to static and dynamic features of these languages. For example, static code analyzers have difficulty to extract facts from C/C++ due to pre-processors, templates and namespaces. The problem is aggregated because embedded applications are developed for the heterogeneous environment and existing tools do not accommodate hardware (eg., ports and interrupts) which are mapped to programming constructs[17]. Furthermore, in these applications the structural and behavioral information is encoded in several distinct languages or technological aspects. The middleware for embedded systems is becoming increasing widespread due to its role in telecommunication networks, military defense and manufacturing automation [31]. Figure 1 shows how an embedded device is intermixing code of C and Java through JNI (Java Native Interface). The reverse engineering community has not paid proper attention to the development of tools which can extract information from such multilingual embedded applications.

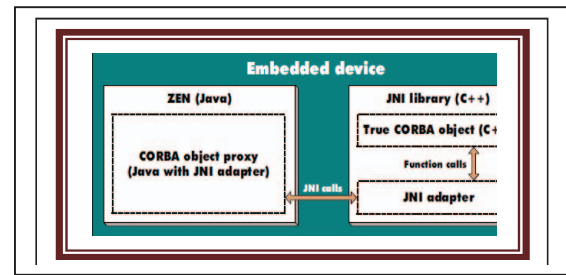


Fig. 1. Wrapping of C and Java Code[15]

Similarly, web applications are developed using different technologies such as Java Scripts, HTML, DHTML, XHTML, XML, CSS, ASP, PHP, DOM, database, images etc. ASP technology has become a favorite tool for developers of web applications as it supports client and server side languages in the same file. It is similar to other technologies such as JSP, ColdFusion and some others for developing dynamic web pages. The single technology based source code parsers are available which support source code parsing of one technology, but support for multiple technologies is missing. The example in Figure 2 presents sample multilingual ASP code which contains code of three languages (Visual Basic, JavaScript and HTML). It is comparatively easy for parsing tools to switch between these code scripts based on their tags, but the challenge is to analyze intertwined code and to filter HTML tags which are not relevant [34]. Similarly, JSP technology combines HTML, XML, JDBC, SQL and other static contents for developing dynamic web pages and analysis of multiple client and server side scripts becomes complicated.

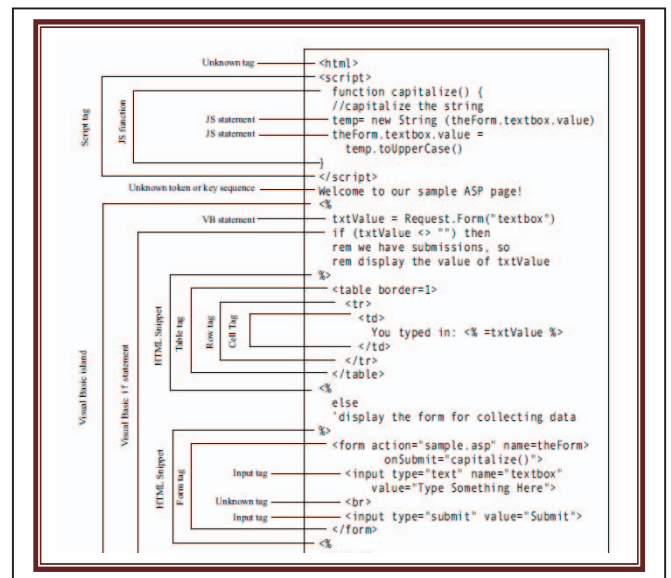
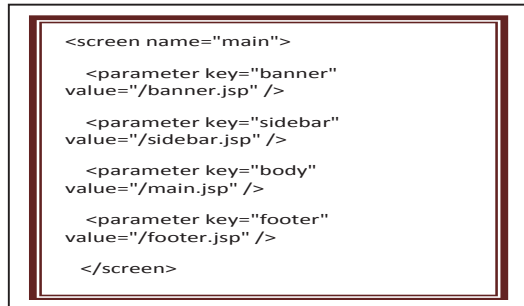


Fig. 2. A Sample of Multilingual ASP Code[34]

The example described in [18] presented in Figure 3 show application of multiple source code elements in Composite view pattern. There are several strategies for applying the Composite View pattern. What all strategies have in common is that sub-views are defined separately and are individually reusable. The layout of a composite view in the Java Pet Store

sample application is controlled by a Java ServerPages™ (JSP™) page called `template.jsp`. The contents of a specific composite view are controlled by another file, `screendefinitions.xml`, that binds JSP pages to named areas of the template. The code block in Figure 3 shows an excerpt from the Java Pet Store sample application website screen definitions file that defines the "main" screen, binding JSP pages to named areas of the template.



```

<screen name="main">
  <parameter key="banner"
    value="/banner.jsp" />
  <parameter key="sidebar"
    value="/sidebar.jsp" />
  <parameter key="body"
    value="/main.jsp" />
  <parameter key="footer"
    value="/footer.jsp" />
</screen>

```

Fig. 3. Composite View Screen

In the multilingual source code applications, the information related to the structure and the behavior of the system is widely spread across various components and languages and the interactions between different application elements could be hidden [29]. In this context, existing source code analysis tools developed for homogeneous applications are not adequate. These existing techniques and tools help to measure quality or provide information about one aspect of the system and they cannot grasp the complexity of multiple languages and environments [29]. Furthermore, following are the key challenges which source code analyzers have to deal with:

A. Dependencies.

The connections between software components are critical in understanding the behavior of an application that helps to perform tasks such as impact analysis [5] and change propagation [30]. Connections among software components are often not directly encoded in the software, but they can be derived by other connections [38]. In enterprise applications, some connections can be derived from relationships between elements in a different domain. For example, connections at the source code level can be used to link user interface components [2]. To understand the nature of derived connections between elements of multiple source code applications, we need to make them explicit. In addition, we also need to expose the information used to derive these connections and make it accessible at multiple levels of abstraction.

B. Persistency.

Data reverse engineering is also equally important with reverse engineering of the source code [11]. A common technology to ensure data persistence is that of relational databases enabling the combination of software and database analysis [9]. It is crucial to provide support for the evolution of multiple source code applications.

C. Technologies.

The specific technologies and frameworks used to implement applications are an invaluable source of information (e.g. the build systems used to transform the source code into deployable components)[32]. Understanding the impact of these technologies on the application could lead to a deeper understanding of the application structure and behavior which ultimately makes the task of source code analysis difficult.

D. Languages and Application logic.

The source code of an application used to develop the application logic is central to a number of analysis tasks [4]. As long as the software applications become more interoperable, the application logic is needed to be implemented with multiple languages simultaneously. To analyze structural and behavioral aspects of multiple source code applications, we need to be able to deal with more than one language at a time.

E. Architecture.

Software architecture is quite helpful in understanding large applications and it supports their evolution [12]. The major challenges of software architecture reconstruction (SAR) are abstracting, identifying and presenting higher level views from lower level and multiple source code information [12].

F. Staged and embedded languages.

The source code analysis becomes arduous when some languages are nested in each other and they are also compiled into other languages [6]. For example, C language is a prime example of a staged language, where source code analysis crumbles.

G. Responsiveness of IDE services.

There are IDE platforms where multilingual type of applications are developed and executed with some constraints. The IDE responsiveness for reverse engineering of multilingual source code applications requires necessary algorithms to produce accurate cross language analysis results.

IV. CONCLUSION

Source code analysis and manipulation provide valuable information about software applications that could be effectively utilized in areas such as architecture extraction, reuse, maintenance, comprehension, software re-engineering, reverse engineering etc. Source code analysis for multilingual applications is undermined area that requires the attention of research community due to its active role in more than 29 domains of software engineering and re-engineering. It is revealed through the literature review that web, embedded, enterprise and other heterogeneous types of software applications demand research in multilingual source code analysis. The development paradigm is shifting from single language and technology to multiple languages and technologies. We believe that research in this area will boost to meet the myriad challenges mentioned in Section III. The

recovery of information from the multilingual type of software applications is difficult due to external commands, files and hidden dependencies of multiple interacting artifacts. The challenging areas found in analyzing these multilingual source code applications are determined in terms of languages and application logic, Architecture, Persistency, Technologies, Dependencies etc. There is a need of new techniques and tools that can cope with complexity and diversity of multilingual applications for extracting required information from the source code of these applications.

REFERENCES

- [1] Arias, T. B. C., Avgeriou, P. and America, P., "Analyzing the actual execution of a large software-intensive system for determining dependencies", *In Proceedings of 15th WCRE*, pp. 49-58, 2008.
- [2] Aryani A., Perin F., Lungu M., Mahmood A. N., and Nierstrasz O., "Can we predict dependencies using domain information", *In Proceedings of the 18th WCRE*, pp. 55-64, 2011.
- [3] Benedusi P., Cimitile A. and De Carlini U., "Reverse engineering process, document production and structure charts". *Journal of Systems and Software*, Vol 19, pp. 225-245, 1992.
- [4] Binkley D., "Source Code Analysis: A Road Map", *In IEEE Future of Software Engineering (FOSE'07)*, pp. 104-119, 2007.
- [5] Bohner S. A and Arnold R.S., "Software Change Impact Analysis", IEEE Computer Society Press, 1996.
- [6] Bravenboer M. and Visser E., "Concrete syntax for objects: domain-specific language embedding and assimilation without restrictions", *In Proceedings of OOPSLA*, pp. 365-38, 2004.
- [7] Canfora G., "New Frontiers of Reverse Engineering", *In Proceedings of Future of Software Engineering (FOSE'07)*, pp. 326-34, 2007.
- [8] Cerulo L., "On the Use of Process Trails to Understand Software Development". PhD thesis, RCOST University of Sannio, Italy, 2006.
- [9] Cleve A., Mens T. and Hainaut J. L., "Data intensive system evolution", *Computer*, Vol 43, pp.110-112, 2010.
- [10] D. L. Moise, K. Wong, H. J. Hoover, and D. Hou., "Reverse engineering scripting language extensions", *In Proceedings of the 14th International Conference on Program Comprehension*, pp. 295-306, 2006.
- [11] Davis, K. H. and Alken, P. H., "Data reverse engineering: A historical survey", *In proceedings of WCRE*, pp. 70-78, 2000.
- [12] Ducasse S. and Pollet D., "Software architecture reconstruction: A process-oriented taxonomy", *IEEE Transactions on Software Engineering*, Vol 35, No 4, pp. 573-591, July 2009.
- [13] Ebert J., Kullbach B., Riediger V. and Winter A., "GUPRO generic understanding of programs, an overview", *Fachberichte Informatik 7-2002*, Universität Koblenz-Landau, 2002.
- [14] Flores E., Barron-Cedeño A., Rosso P. and Moreno L., "Towards the Detection of Cross-Language Source Code Reuse", *Springer-Verlag Berlin Heidelberg*, pp. 250-253, 2011.
- [15] Giuseppe A. Di L., Anna R. F., F. Pace, Porfirio T., and Ugo de C., "WARE: A Tool for the Reverse Engineering of Web Applications", *In Proceedings of CSMR*, pp. 241-250, 2002.
- [16] Harman M., "Why Source Code Analysis and Manipulation Will Always Be Important", *Tenth IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 7-19, 2010.
- [17] Holger M. Kienle, Johan Kraft and Hausi A. Müller, "Software Reverse Engineering in the Domain of Complex Embedded Systems", *Reverse Engineering Recent Advances and Applications*, (www.intechopen.com)
- [18] Java pet store, "J2EE Composite View Pattern", <http://192.9.162.55/blueprints/patterns/CompositeView.html>. [accessed on 20-02-2014]
- [19] Jawawi D., Deris S. and Mamat R., Software Reuse for Mobile Robot Applications Through Analysis Patterns, *The International Arab Journal of Information Technology*, Vol. 4, No. 3, pp. 220-228, July 2007
- [20] Kullbach B., Winter A., Dahm P. and Ebert J., "Program comprehension in multi-language systems", *In Proceedings of WCRE*, pp. 135, 1998.
- [21] Linos P., Lucas W., Myers S. and Maier E., "A metrics tool for multi-language software", *In Proceedings of the 11th IASTED International Conference on Software Engineering and Applications, SEA '07*, pp. 324-329, 2007.
- [22] Burnett, M., Cook, C. and Rothermel, G., "End-user software engineering", *Commun. ACM*, Vol 47, No. 9, pp.53-58, 2004.
- [23] Marchetto A., Tonella P. and Ricca F., "ReAjax: a reverse engineering tool for Ajax Web applications", *IET Software* Vol 6, No 1, pp. 33-49, 2012.
- [24] Marinescu C. and Jurca I., "A meta-model for enterprise applications", *In Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 187-194, 2006.
- [25] Marinescu C., "Discovering the objectual meaning of foreign key constraints in enterprise applications", *In Proceedings of 14th WCRE*, pp. 100-109, 2007.
- [26] Marinescu C., "Identification of Relational Discrepancies between Database Schemas and Source-Code in Enterprise Applications", *In Symbolic and Numeric Algorithms for Scientific Computing, SYNASC*, pp. 93-100, 2007.
- [27] MoDISCO Eclipse, <http://www.eclipse.org/MoDisco/> [accessed on 12-2-2014]
- [28] Object Management Group, Meta object facility (MOF) 2.0 core final adopted specification. *Technical report, Object Management Group*, 2004.
- [29] Perin F., "Reverse Engineering Heterogeneous Applications", *PhD Thesis, Institut für Informatik und angewandte Mathematik*, 2012.
- [30] Rajlich V., "A model for change propagation based on graph rewriting", *In Proceedings of the International Conference on Software Maintenance, ICSM'97*, pp. 84-91, 1997.
- [31] Schirmer G., Harmon T. and Klefstad R., "Late Demarshalling: A Technique for Efficient Multi-language Middleware for Embedded Systems", *LNCS*, Vol 3291, pp. 1155-1172, 2004.
- [32] Spinellis D., "Software builders", *IEEE Software Journal*, Vol 25, No 3, pp. 22-23, May 2008.
- [33] Strein D., Kratz H. and Loewe W., "Cross-language program analysis and refactoring", *In Proceedings of the International Workshop on Source Code Analysis and Manipulation*, pp. 207-216, 2006.
- [34] Synytskyy N., Cordy R., J., Dean R. T., "Robust Multilingual Parsing Using Island Grammars", *In Proceedings of the conference of the Centre for Advanced Studies on Collaborative research (CASCON '03)*, pp. 266-278, 2003.
- [35] Terceiro A., Costa J., Miranda J., Meirelles P., Rom'ario R. L., Almeida L., Chavez, C. and Kon F., "Analizo: an extensible multi-language source code analysis and visualization toolkit, *Brazilian Conference on Software: Theory and Practice*, Vol 29, 2010.
- [36] Tichelaar S., Ducasse S., Demeyer S. and Nierstrasz O., "A meta-model for language-independent refactoring", *In Proceedings of International Symposium on Principles of Software Evolution*, pp. 157-167, 2000.
- [37] Voelter M., Ratiu D., Kolb B. and Schaezt B., "mbeddr: Instantiating a language workbench in the embedded software domain", *Journal of Automated Software Engineering*, Vol 20, pp.339-390, 2013.
- [38] Yu Z. and Rajlich V., "Hidden dependencies in program comprehension and change propagation", *In Proceedings of the 9th International Workshop on Program Comprehension*, pp. 293-299, 2001.
- [39] Zaroni M., "Data mining techniques for design pattern detection", PhD Thesis (Tesi di dottorato, Università degli Studi di Milano-Bicocca, 2012).
- [40] Klint, P., Van Der Storm, T. and Vinju, J., "Rascal: A domain specific language for source code analysis and manipulation", *In Proceedings of Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 168-177, 2009.