Detection of J2EE Patterns based on Customizable Features

Zaigham Mushtaq, Ghulam Rasool, Balawal Shahzad
COMSATS Institute of Information Technology, Lahore

***Abstract*—Design patterns support extraction of design information for better program understanding, reusability and reengineering. With the advent of contemporary applications, the extraction of design information has become quite complex and challenging. These applications are multilingual in nature i.e. their design information is spread across various language components that are interlinked with each other. At present, no approach is available that is capable to extract design information of multilingual applications by using design patterns. This paper lays foundation for the analysis of multilingual source code for the detection of J2EE Patterns. J2EE Patterns provide design solutions for effective enterprise applications. A novel approach is presented for the detection of J2EE Patterns from multilingual source code of J2EE applications. For this purpose, customizable and reusable feature types are presented as definitions of J2EE Patterns catalogue. A prototype implementation is evaluated on a corpus that contains the repository of multilingual source code of J2EE Patterns. Additionally, the tool is tested on open source applications. The accuracy of the tool is validated by successfully recognizing J2EE Patterns from the multilingual source code. The results demonstrate the significance of customizable definitions of J2EE Pattern's catalogue and capability of prototype.**

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

I. INTRODUCTION

Design patterns are verified solutions that provide solid foundation for the development of effective software applications [1, 2].Every design pattern has its own intent and particular aspect. Accurately recovered design patterns helps to understand the structure and behavior of the application [3-7]. Therefore, they can be used in better program understanding, reverse engineering, reengineering and refactoring[4-6, 8-14].

Modern applications are essential part of our daily life. They are all around us from navigational systems to medical equipment. These contemporary applications are heterogeneous in nature and composed of multiple source code languages. They are present in the form of embedded systems, enterprise applications (J2EE environment), mobile applications and web based applications etc. The analysis of multilingual applications is difficult and challenging due to the presence of multi-language artifacts, external files and hidden dependencies of multiple interacting components [4]. Moreover, the recovery of cross language artifacts is hard as most of the program comprehension approaches focus on extracting information from homogeneous applications. The existing approaches do not provide generic and extensible solutions to support multilingual applications.

Java Enterprise applications are one of the examples of distributed multilingual applications. This environment contains multiple language components such as JavaBeans (EJBs), JSPs, Servlets etc. The analysis of J2EE application is difficult and challenging due to the following reasons.

- J2EE applications are multitier applications i.e. the software components fall across different layers. The information is scattered across various components and sources. In order to get the structure of the application we have to deal with all the layers.
- J2EE applications are difficult to analyze because of the presence of cross language artifacts. These artifacts are built in multiple languages including Java, JSP, HTML, XML, SQL etc. There is heterogeneity across language boundaries and the information is distributed in cross language artifacts that are interdependent with each other. The cross language artifacts interact with each other to perform a particular task. They may have hidden dependencies. It is very difficult to resolve these cross language artifacts (XLAs) and extract architectural details.

J2EE environment is equipped with proven solutions in the form of J2EE Patterns. They help in building flexible enterprise applications [13]. The importance of J2EE Patterns cannot be ignored in terms of its recovery and reusability. Following points characterize the significance of J2EE Patterns.

- J2EE Patterns expose the design and intent of multilingual applications. Their recovery helps in identification of key aspects of common design structures.
- Recovery and utilization of design is beneficial in minimizing work effort in terms of maintenance, development and investment cost, brings improvement in software security and design consistency.
- J2EE Patterns help to improve software quality. Their reusability supports maintainable, simple, and clean enterprise applications [2].
- The utilization of J2EE Patterns enhances design vocabulary and allow to build an application at higher level of abstraction.

Patterns of Java enterprise applications (JEAs) or J2EE Patterns are described to build an effective enterprise application [5, 6, 8]. In order to build an effective analyzer, it is

necessary to completely define the features of the J2EE design patterns. The prototype model should analyze the source code on the basis of definition and features of J2EE Patterns and recognize these patterns from source code of multiple languages (Java, JSP, Servlets, SQL etc.).

A complete definition of J2EE Patterns is required for effective detection and analysis of multilingual enterprise applications. There are no specifications or definitions available for the detection and recovery of J2EE Patterns. Definitions and features are required for the accuracy and flexibility of feature is the key in recognition of patterns. The expected model shall incorporate the complete features.

The recovery of J2EE Patterns is challenging due to the following reasons.

- J2EE Patterns have abstract representations and usually their documentation is not available in the source code. The instances of J2EE Patterns are scattered in different Languages and there is no formal rule of their composition.
- As far as the recognition of J2EE Patterns is concerned, to the best of our knowledge there is no approach or tool available which is capable to detect J2EE pattern from multilingual source code of enterprise applications.
- There is no benchmark system available for comparison and validation of results of J2EE Patterns.

In this paper, enhanced semi-formal definitions of J2EE Patterns are presented in the form of customizable and reusable feature types. These feature types cover the aspects to redefine J2EE Patterns in the form of inheritance, composition, delegation, association and cross language links (XLLs) etc. A novel approach is presented for the recovery of J2EE Patterns from multilingual source code of enterprise applications. Initially pattern's definitions are extracted from standard resources of J2EE Patterns [5, 6, 8, 10, 15, 16]. On the basis of these definitions, features types are developed.

Following objectives have been achieved as major contributions.

- First of all, fundamental properties of J2EE Patterns are extracted from reliable and authentic resources.
- On the basis of J2EE Pattern's properties, customizable and reusable feature types are created. The capability of features can be enhanced easily. Moreover, new patterns can be included simply in the existing features. The presence of features enhances the adaptability in improved catalogue of J2EE Patterns.
- A catalogue of J2EE Pattern's definitions is created by using customizable feature types. Customizable and adaptable features allow to add new pattern definition or accommodate their variants.
- A prototype is developed as a plugin with Visual Studio.Net framework using Sparx Enterprise Architect Modeling Tool. This tool uses pattern definitions and is capable to recover J2EE Patterns. From multilingual source code of Java Enterprise Applications.
- A corpus is built that contains the repository of source code of J2EE Pattern definitions from reliable resources [5, 6, 8-10, 15-17]. This repository is used to test the validity of the prototype. This prototype is also evaluated on open source code J2EE applications.

This paper is organized as, Section II contains related work of design pattern detection approach, Section III describe J2EE Pattern's definitions, Section IV presents J2EE Pattern's Feature, Section V contains Pattern representation in terms of Feature Types, Section VI describes process of Feature Types extraction and pattern recognition and Section VII presents conclusion and future research.

II. Related Work

Design patterns recognition promotes extraction of architectural details and design decisions from source code. Recovering pattern instances supports program comprehension and helps to adapt applications to meet with the current and future requirements. A number of different design pattern detection approaches are proposed in the literature. Some of the important tools and approaches are presented in this section.

Coppel et al. [18], presented a deprogramming approach for architectural understanding of large and complex software applications. Deprograming is a process to recover concept, design and patterns from source code. They proposed a tool DeP that translates source code into dependency graph and then mines through the design patterns. This tool supports design pattern detection, code smell detection and automated source code documentation.

Costagliola et al. [19], presented a visual language based tool for the recovery of design patterns. They followed a two phase model. The 1st phase involves recovery of design instances using coarse grained analysis and in 2nd phase, design patterns are recognized by using fine grained analysis. They use UML class diagram that is mapped on language grammar for design pattern detection. The proposed tool focused on structural aspects of design patterns. However, the proposed tool suffers from scalability issues and disparity in design pattern recovery.

Dong et al. [20], presented a toolkit DP-Miner that recover design patterns from source code by following weight and metrics criteria. They inspect the source code by providing structure and design pattern descriptions in the form of a metrics. This tool however, has limited precision and recall.

O. Kaczor et al. [21], presented a reverse engineering tool, PTIDEJ, for the analysis and maintenance of object oriented applications. This tool performs pattern trace identification and enhancement in object oriented software. This tool performs model analysis by using PADL Meta model (Pattern abstract and level description language). The results of presented approach show maximum recall, however, the precision of pattern recovery is compromised.

N. Shi et al. [7], developed an automated design pattern recognition tool, PINOT. PINOT is built on an open source

compiler, Jikes along with a pattern analysis engine. This tool is capable to recognize structural and behavioral aspects of design patterns. Although this tool recognizes all GOF patterns, the main drawback of PINOT is that it is not customizable for new or extended data structures.

Fontana et al. [22], presented a tool, MARPLE as an eclipse plugin for design patterns extraction and software architecture reconstruction. This tool is designed to be language independent, however, currently it available for Java.

G. Rasool et al. [3], presented an approach for design pattern detection that is based on variable feature types. They use multiple search techniques that allow the flexible detection of design pattern variants. The results ensure the accuracy of the approach in successful recognition of design patterns as compared to previously presented techniques. The idea of using customizable feature types for pattern detection is quite effective for handling variants of design patterns. We used this concept for providing customizable and flexible definitions of J2EE patterns.

Concluding from the discussion of related work is that the proposed approaches are focused towards the detection of GOF Patterns and suitable for homogeneous applications. There is no approach capable to detect design patterns from multilingual applications. For example, the intent of J2EE Patterns is implemented in multiple languages, including Java, JSP, Java Beans, Servlets, SQL etc. In-order to recover J2EE Patterns, complete understanding of multi-language artifacts is required that participate in pattern's definition. Moreover, the cross language artifacts may have hidden dependencies. Therefore, in-order to recover architectural information of J2EE Patterns, we need to resolve the complexity and inter-dependency of cross language components.

III. Process of Creating Pattern's Definitions and Feature Types

In this section, complete detail in the form of different aspects of J2EE patterns is presented. J2EE Pattern's definition covers the prospects in terms of implementation and reverse engineering details. The implementation perspective ensures the most common and necessary components that are required for implementing the desired patterns. Whereas, the reverse engineering approach ensures the successful recovery of J2EE Patterns from the enterprise applications. This aspect covers the features comprising the definition of the requisite pattern that needs to be extracted from source code.

In this research each and every aspect of the J2EE Patterns is discussed that provide the basis for building pattern detection criteria. These aspects include definition, description, components, roles and features. The standard definitions of J2EE Patterns are extracted from quality resources [5, 6, 8, 10, 15, 16]. These definitions are presented in the form of extendable and reusable features, which can be translated in the form of multiple techniques and algorithms. These features can be used for further enhancement and detection of other patterns.

Once a pattern definition is completed; it is then added to the pattern definition catalogue. The pattern search engine can get a pattern definition one by one and execute a search by traversing feature types in each of pattern's component's definition.

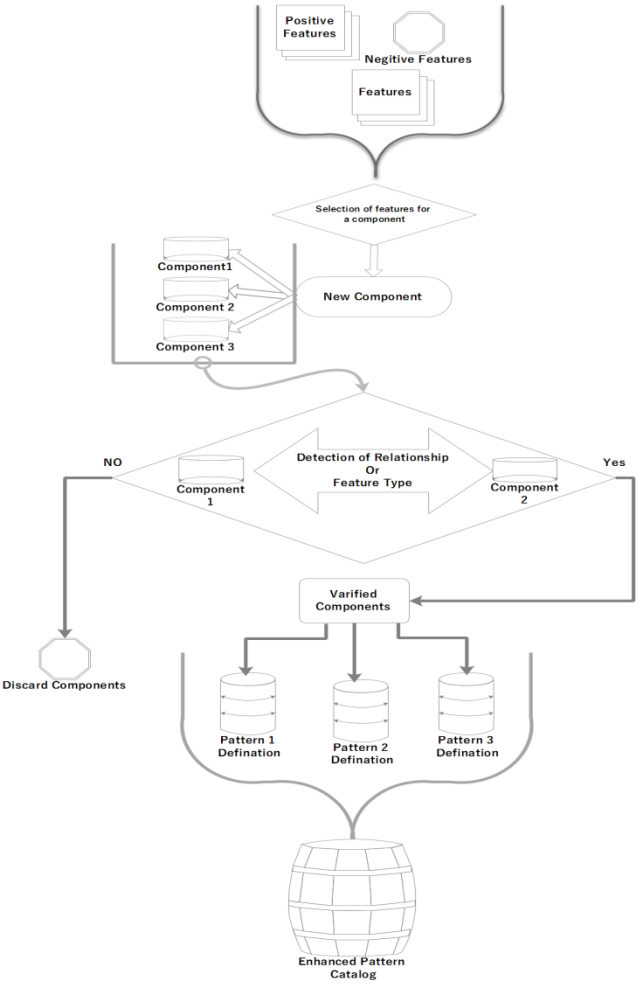

Fig. 1. Pattern Definition Approach

IV. Definitions of J2EE Patterns

Features are building block for a pattern. A pattern is a combination of multiple features that is implemented in the source code. These features are helpful for the development and recovery of patterns from source code.

In this section, multiple components of J2EE Patterns are identified and then one or more features are built together to define a pattern's instance. A pattern is defined by its components and relationship between them. First of all, multiple components of J2EE Patterns are identified and then for each component multiple feature types are added. After defining two or more components, relationships (Feature type) between those components are specified.

A. Data Access Object (DAO) Pattern

- DAO pattern detach data accessing API from client or business object, it separates domain logic to communicate with database by introducing data access layer between business object and Database.
- It decouples persistence storage implementation from rest of the application.

- DAO layer is responsible for data access from persistence storage and manipulates data in persistence storage.

TABLE I. FEATURES OF DATA ACCESS OBJECT PATTERN

Index	F #	Feature's Signature
PF1	F1	HasClassesWithGeneralizations (AllObjs)
PF2	F2	HasCRUDOperations (PF1)
PF3	F3	HasConnectionString (AllObjs) OR HasDataSource (AllObjs)
PF4	F35	HasNoCRUDOperation (PF3)
PF5	F5	HasAssociation (PF1, PF3)
PF6	F6	HasDTOs (AllObjs)
PF7	F5	HasAssociation (PF5, PF6)
PF8	F5	HasAssociation (PF7, AllObjs)

B. Data Transfer Object (DTO) Pattern

DTO is Only a Data Buffer to reduce the remote procedure calls (RPCs) on data Access layer and thus reducing the network traffic.

TABLE II. FEATURES OF DATA TRANSFER OBJECT PATTERN

Index	F #	Feature's Signature
PF1	F7	Count (AllObjs, Methods)
PF2	F8	Count (AllObjs, Attributes)
PF3	F9	IsTrue (AllObjs, PF1>=PF2)
PF4	F36	HasNotMethodsCount (PF3, <0)
PF5	F37	HasNotAttributesCount (PF4, <0)
PF6	F10	HasGettersCount (PF5, >=PF2)
PF7	F11	HasSettersCount (PF6, >0)
PF8	F38	HasNotDefinedType (PF7, "DataSource"||"Connection")
PF9	F35	HasNoCRUDOperation (PF8)
PF10	F5	HasAssociation (PF8, AllObjs)

C. Value Object (VO) Pattern

Value object is Only a Data Buffer to reduce the RPCs on data Access layer and thus reducing the network traffic. The difference between DTO and VO is that VO are immutable, as they do not allow change once created, they are read only. Thus they don't provide setter Functions.

TABLE III. FEATURES OF VALUE OBJECT PATTERN

Index	F #	Feature's Signature
PF1	F7	Count (AllObjs, Methods)
PF2	F8	Count (AllObjs, Attributes)
PF3	F9	IsTrue (AllObjs, PF1>=PF2)
PF4	F36	HasNotMethodsCount (PF3, <0)
PF5	F37	HasNotAttributesCount (PF4,<0)
PF6	F10	HasGettersCount (PF5, >=PF2)
PF7	F11	HasSettersCount (PF6, <0)
PF8	F38	HasNotDefinedAType (PF7, DataSource"||"Connection")

D. Service Locator (SL) Pattern

This pattern is used to locate JMS or EJB services by JNDI registry service lookup. This pattern uses context object to locate requisite service and cache (object) mechanism to reduce cost of JNDI lookup.

TABLE IV. FEATURES OF SERVICE LOCATOR PATTERN

Index	F #	Feature's Signature
Service		
PF1	F12	GetAllInterfaces ()
PF2	F13	HasClassWithGeneralizations (AllObjs) //Candidate Service Objects
Service Locator		
PF3	F5	HasAssociation (PF2, PF1)
PF4	F14	HasMethodWithRType (PF3, PF1|"Object")
PF5	F15	HasMethodWithParameterType (PF3, PF2|"String")
Initial Context		
PF6	F5 | F39 & F14 & F14	(HasAssociation (F5, F5, Where (PF5! = PF5)) OR HasNoMethodWithParameterType (PF5, PF2)) AND HasMethodWithRType (PF3, PF1) AND HasMethodWithRType (PF3, PF1)
PF7	F5	HasAssociation (PF5, PF6) OR HasMethodWithParameterType (PF6, PF1)
Cache Objects		
PF8	F15	HasMethodWithParameterType (PF6, PF2)
PF9	F5	HasAssociation (PF5, PF8)

E. Value List Handle (VLH) Pattern

A value list handler pattern provides an efficient way to iteratively manage a large set of data in the form of a read only list of values. This pattern helps the client to iterate through collection of results populated in list of user interface.

TABLE V. FEATURES OF VALUE LIST HANDLER PATTERN

Index	F #	Feature's Signature
PF1	F17	GetAllDtos ()
PF2	F5	HasAssociation (PF1, AllObjs)
PF3	F18	HasGeneralization (PF2, AllObjs)
PF4	F19	HasRealization (PF3, AllObjs)
PF5	F19	HasRealization (PF5)
PF6	F20	HasDefinedAType (PF5,"Itrator"|"List")
PF7	F21	HasMethodNameWhichContains (PF6,"Next"| "Previous")
PF8	F22	GetAbstractClasses ()
PF9	F18	HasGeneralization (PF7, PF8)

F. Front Controller (FC) Pattern

The front controller pattern is a single controller that handles all the requests for a web application. This pattern provides centralized request handling mechanism and act as entry point for all requests coming to the web application.

TABLE VI. FEATURES OF FRONT CONTROLLER PATTERN

Index	F.#	Feature's Signature
Helpers		
PF1	F20	HasDefinedAType (AllObjs, "Dispatch")
PF2	F40	HasNoRealizationWithType (PF1, "HttpServlet")
Front Controllers		
PF3	F18	HasGeneralization (AllObjs, "HttpServlet")
PF4	F21	HasMethodNameWhichContains (PF2,"doGet" | "doPost")
PF5	F5	HasAssociation (PF3, PF2)
PF6	F23	HasDelegation (PF5, PF2)

G. Session Façade (SF) Pattern

The session façade is implemented as a session bean that exists at higher level and connected with the lower level business tier components. The lower level components could be entity bean, session bean or DAO. This pattern serves as layer that wraps the lower level business components. The

client could only access the methods of business components only though the session bean.

TABLE VII. FEATURES OF SESSION FACADE PATTERN

Index	F.#	Feature's Signature
PF1	F42| F24	HasRealizationWithType (AllObjs, "javax.ejb.SessionBean") OR HasAnnotation (AllObjs," @stateless" | "@stateful")
PF2	F20| F14& F43	HasDefinedAType (F1, GetDAO ()) OR HasMethodWithRType (F1, GetDTO()) AND HasMethodContainsDelegation (F1,GetAllMethods (F1))
PF4	F27	HasMethodLineOfCode (FP3, <= F3)
Business Object		
PF5	F28	GetAllClasses ()
PF6	F5	HasAssociation (F5, F6)

H. Business Delegate (BD) Pattern

Business Delegate serves as layer between client and business service. This layer is responsible for accessing business service methods using lookup service. This pattern decouples presentation tier from business tier. Business Delegate pattern is responsible to hide business service detail from client such as lookup and access mechanism. In order to provide access to business services, the business delegate use lookup service to business service.

TABLE VIII. FEATURES OF BUSINESS DELEGATE PATTERN

Index	F.#	Feature's Signature
Business Lookup		
PF1	F12	GetAllInterfaces ()
PF2	F5	HasAssociation (AllObjs, F1)
PF3	F15	HasMethodWithParameterType (AllObjs, F2| "Object"| "String")
PF4	F14	HasMethodWithRType (F3, F2| "Object"| "String"| "T")
PF5	F41	HasNoDelegation (F4, F2)
Business Delegate		
PF6	F28	GetAllClasses ()
PF7	F15	HasMethodWithParameterType (F6,"String"| "string")
PF8	F39	HasNoMethodWithParameterType (F7, F1)
PF9	F23	HasDelegation (F8, F5)
Service		
PF10	F19	HasRealization (AllObjs, F2)
Client		
PF11	F23	HasDelegation (AllObjs, F9)

I. Composite view (CV) Pattern

A composite view pattern allows a parent view to aggregate sub views so that overall view becomes a combination of small atomic parts. We can create composite view from multiple atomic sub views. Composite view is actually used for separating and managing layout from the actual contents.

TABLE IX. FEATURES OF COMPOSITE VIEW PATTERN

Index	F.#	Feature's Signature
Mapper		
PF1	F29	GetXMlObjects ()
PF2	F30	HasNumberOfAssociationsWithType (F1,>=2, "HTML" | "JSP")
PF3	F31	HasTheseXMLTags (F2, "Include"| "Put")
Template		
PF4	F32	GetJSPObjects ()
PF5	F33	GetHTMLObjects ()
PF6	F30	HasNumberOfAssociationsWithType (F4, >=1, "HTML" | "JSP")
PF7	F5	HasAssociation (F5, F3)
View		
PF8	F34	HasNoNumberOfAssociationsWithType (F4 >=1, "HTML" | "JSP")
PF9	F5	HasAssociation (F7, F3)

J. Intercepting Filter

An intercepting filter offers pluggable filters, providing common services for preprocessing incoming client requests and post processing the responses.

TABLE X. FEATURES OF INTERCEPTING FILTER PATTERN

Index	F.#	Feature's Signature
Filter Chain		
PF1	F28	GetAllClasses
PF2	F30	HasNumberOfAssociationsWithType(1, ("Class"&&"Interface"),PF1)
Filter		
PF3	F32	GetAllAssocationsOfObject(PF2)
PF4	F19	HasRealization (PF3)
Filter Manager		
PF5	F23	HasDelegation (AllObjs,PF4)
Client		
PF6	F23	HasDelegation(AllObjs,PF5)

V. CATALOGUE OF FEATURE TYPES OF J2EE PATTERNS

In this section 43 different feature types of J2EE patterns are presented. The feature types include 35 positive features (Table 1) and 8 negative features (Table 2). Our technique exploits reusable feature types and utilizes them to characterize pattern definitions. Predefined feature types are provided as a catalogue and the user is allowed to create its own definition by selecting multiple features to recognize a specific pattern. Our Catalogue of features is also easily extendable. Following characteristics are observed during the process of creating J2EE features/patterns.

- JEE Patterns are assembled by using object oriented features.
- Programming language construct are used to elaborate the features.
- Generic parameters are used which are implementable in multiple languages.
- The features are extendable and customizable to accommodate new patterns or to adapt with any variation.

Feature types are mentioned and described comprehensively in Table 2.

Negative Features: Negative feature types are used to negate the specific characteristics of the patterns. During pattern's detection process the negative feature types help to reduce false positives.

J2EE Patterns by Using Feature Types: In this section J2EE Patterns are redefined on the basis of feature types (mentioned in previous section). These patterns can be represented as a combination of feature types.

TABLE XI. FEATURE TYPES OF J2EE PATTERNS

F. #	Feature types	Description
F1	HasClassesWithGeneralizations (AllObjs)	This feature returns all the classes which are inherited from at least one object (class or interface) or API.
F2	HasCRUDOperations (C)	This Feature takes all the classes from source code as a parameter and returns all those classes which contain SQL based CRUD (create, read, update, delete) operation.
F3	HasConnectionString (C)	This Feature returns all those classes which define a connection string with database.
F4	HasDataSource (C)	This Feature returns all those classes which define object of Data Source type.
F5	HasAssociation (C1,C2)	This Feature returns Boolean expression (true) if class, C1 has an association with class, C2, e.g. C1 create s C2 inside its code.
F6	HasDTOs (AllObjs)	This feature accepts all the objects as a parameter and returns only those classes which have defined only attributes and provides getter and setter methods for these attributes.
F7	Count (Obj, Methods)	This feature returns method count from a given object.
F8	Count (C, Attributes)	This feature returns attribute count from a given class.
F9	IsTrue (AllObjs, F1>=F2)	This feature accepts all the objects and count of features from F or F2 and returns all those objects which are matched with the given condition
F10	Has GettersCount (AllObjs, Condition Expr)	This feature returns classes with Getter methods count matched with the specified conditional expression.
F11	HasSettersCount (ALLObjs, >X)	This feature returns classes with Setter methods count matched with the specified conditional expression.
F12	GetAllInterfaces ()	This feature returns all Interfaces from source code.
F13	HasClassWithGeneralizations (C)	Returns Classes from C which have inheritance relationship
F14	HasMethodWithRType (C1, C2|"Object")	This feature returns only those Classes from C1, whose method's type is matched with Classes C2's methods or method's return type matches with 'string' or 'object'.
F15	HasMethodWithParameterType (C1,C2|"String"|"Object")	This feature returns only those classes from C1 whose methods parameter's type is matched with Classes of C2's method's parameter types or method parameter type matches 'string' or "object".
F16	GetCacheObjects ()	This feature returns all classes which can cache another class.
F17	GetAllDtos ()	This feature returns all Classes that can act as a data transfer object
F18	HasGeneralization (C1,C2)	This feature returns classes from C1, in case if classes in C1 have Generalization with classes in C2
F19	HasRealization (Objs1,Objs2)	This feature returns objects from Obj1, if Objs1 has Generalization with Objs2
F20	HasDefinedAType (C1,T = "Iterator"|"List")	This feature returns Classes, if class C1 matches with type name in T e.g. (iterator or list).
F21	HasMethodNameWhichContains (C, M = "Next" | "Previous")	This feature returns Classes, if class C matches with Method Name in M e.g. (Next or Previous)
F22	GetAbstractClasses ()	This feature returns all abstract classes from source code.
F23	HasDelegation (C1, C2)	This feature returns Boolean expression "True", if class C1 calls class C2.
F24	HasAnnotation (AllObjs, " @stateless" | "@stateful")	This feature filters out all those object which don't have the annotations (stateless or stateful)
F25	MethodsContainsDelegation (GetAllMethods (C1))	This feature returns Boolean expression "True", all methods of class C1 contains a Delegation.
F26	HaMethodsCount (C1, Condition Expr = " <=X")	This feature returns all those classes from C1 which have method count matches with condition X.
F27	HasMethodLineOfCode (C1, Condition Expr = "<=Y")	This feature returns all those classes from C1 which have method Line count matches with the condition Y.
F28	GetAllClasses ()	This feature returns all classes from source code
F29	GetXMlObjects ()	This feature returns all XML objects from source code.
F30	HasNumberOfAssociationsWithType (Obj1, Condition Expr =">="X, T="HTML"|"JSP")	This feature returns all those Objects which have number of Association given in Condition Expression and object Type provided in T.
F31	HasTheseXMLTags(Objs1, TG ="Include"|"Put")	This feature returns objects in Objs1 contain Tags given in TG.
F32	GetJSPObjects ()	This feature returns all objects of type JSP from the source code.
F33	GetHTMLObjects ()	This feature returns all objects of type HTML from the source code.
F42	HasRealizationWithType (F1,"HttpServlet")	This feature returns classes from F1 which implement a specific interface
F43	HasMethodContainsDelegation (GetAllMethods (C1))	This feature returns Boolean value true, if given methods of C1 contains a delegation.

TABLE XII. NEGATIVE FEATURE TYPES

F. #	Negative feature types	Description
F34	HasNoNumberOfAssociationsWithType (F4 >= X, "HTML" | "JSP")	This feature returns all those Objects which will not have number of Association given in Condition Expression and object Type provided in T.
F35	HasNoCRUDOperation (F3)	This Feature takes all the classes from source code as a parameter and returns all those classes which will not contain SQL based CRUD (create, read, update, delete) operation
F36	HasNotMethodsCount (F3, < X)	This feature returns all those classes from C1 which do not have method count according to the provided condition.
F37	HasNotAttributesCount (F4, < X)	This feature returns all those classes from C1 which do not have Attribute count according to the condition provided.
F38	HasNotDefinedAType (F7, T =" DataSource" ||" Connection")	This feature returns Classes, if classes return by F7 Do not matches with type name in T (Data Source or Connection).
F39	HasNoMethodWithParameterType (C1, C2)	This feature returns only those methods from classes C1 whose parameter's type is Not matched with Class C2.
F40	HasNoRealizationWithType (F1," HttpServlet")	This feature with return those classes from F1 which will have implemented a specific interface
F41	HasNoDelegation (F4, F2)	All those classes from F4 which do not have any delegation to classes in F2

VI. J2EE PATTERN'S RECOGNITION APPROACH

J2EE patterns are presented by Sun Micro Systems [6, 8, 23]. These patterns exhibit interclass relationships like GOF patterns. J2EE patterns uses GOF patterns [7] as their base, however, they support multiple and different language components.

In-order to recover J2EE Patterns from source code, the user needs to build a pattern definition by using different feature types in the form of pattern's catalogue. It is necessary to understand feature types and how they are used in pattern's definition. For this purpose, source code of J2EE Patterns is analyzed from authentic resources [5, 6, 8-10, 15-17] and then necessary components and relationships between these components are realized.

The feature extraction and pattern recognition approach exploits object oriented classes (abstract classes, concrete classes, and interface classes etc.) and interclass relationships (inheritance, association, realization, delegation etc.). In order to get precise recognition of interclass relationships [24] between pattern's components, some filters are also applied in the form of negative feature types that filters out false negatives. Successful mining of these relationships helps in detection of pattern's instances.

A. Explaination of The Approach

The feature types cannot be compared directly from source code. First of all, the source code is transformed and abstracted into relational database model (RDB Model). The Enterprise Architect Tool is used to abstract the source code into relational data model (RDB Model). The main advantage of using RDB model is that we can execute any SQL statement easily. Following steps transform source code in a proper intermediate representation.

1) Use of Source Code:

Our approach uses source code for further processing, not the binary data.

2) Creating Initial model using Enterprise Architect (EA):

In the first step, Enterprise Architect (EA) recovers source code of multiple languages into its RDB model one by one. The RDB model is an initial model which contains abstract information of parsed source code. This model contains a rich set of R-Tables which encapsulate many possible aspects of source code. This information is further used by J2EE Pattern Detection (JPDT) Tool. This model has following major tables of our concern, which contains abstract information about the parsed source code.

a) t_object Table: In this table main entities from source code are parsed. For an example, if Java source code is parsed, classes and interfaces are stored in this table. This table is connected with all other table in model's schema, by defining t_object. Some most important attributes of this table are explained in the form of Objetc_ID, Object_Type, Name, Abstract, GenType, GenFile and GenLinks.

b) t_connector Table: This table contains all types object oriented relations identified by EA tool. However, capability of EA is limited and cannot completely resolve all type of relationships (limitation of EA is discussed in later section). For example, if Class A is inherited from Class B then t_connector retains that information which notifies that Class A is inherited from Class B. This Information is presented in the form of a Start_object_ID as (Class A) and End_Object_Id as (Class B).

c) t_operations Table: This table contains information about methods or procedures duly parsed from source code. Since many languages support procedural programming, therefore all EA's supported languages contain procedures in same format.

d) t_Attribute: This table contains all the class level member of a GPL which support object oriented concepts, attributes are those variables which are created with class level access.

e) t_operationParams: Method/Procedures are called by passing them parameters. This table contains all parameter along with their reference to the related method and with their type and name.

f) t_package: This table contains all the packages /namespaces in a source's base line.

3) Limitations of Enterprise Architect (EA) Tool:

Enterprise Architect (EA) is a powerful modeling tool. This tool incorporates plethora of information and supports the analysis of source code applications by providing easy SQL statements. However, EA tool is deficient to analyze multilingual applications; their mechanism only supports the query of single source code applications. Enterprise applications like J2EE/ JAE applications are composed of multiple language artifacts. These artifacts interact with each other across language boundaries by using different cross language associations. Therefore, in-order to analyze multilingual application (like J2EE applications) we need to extend the basic enterprise architect model so that multilingual aspects can be extracted by executing SQL statements. Enterprise Architect (EA) however has deficiencies in basic model of EA tool. This tool has following limitations.

a) EA can only parse single source code applications. e.g. Java, Python, C# etc.

b) EA do not support web based languages like ASP.Net, HTML, JSP, PHP etc.

c) EA cannot parse Domain Specific Languages (DSLs).

d) EA can only extract strong inter-class relationships. e.g. EA can detect only class level association, like a class defining an attribute of another class.

e) EA cannot extract cross language relationship at any level, which in our case is required for the detection of J2EE Patterns.

f) EA cannot resolve relationships slightly complicated relationship like Delegation relationship, uses relationship etc.

g) EA is deficient to support queries to extract links between Cross language artifacts e.g. how one artifact refers to other or how one artifact is referred in other artifact.

h) In initial model there is a limited support to extract association relationships between artifacts. EA do not support weaker forms of relationships (associations) between the artifacts of general purpose languages (GPLs). e.g. A class defining an object of another class in a function's body or in a function's parameter. EA is deficient to detect

- Delegation between artifacts of multiple languages;
- Associations through local variables;
- Associations through function's parameters;
- Associations through function return type;
- Associations between cross language components;
- Other forms of associations like aggregation.

4) Extended Super Model (JPSP), an Enhancement of Enterpsise Architect Model:

Enterprise Architect is a well-recognized UML based modeling tool for design and development of software system [25-29]. This tool is capable to reverse engineer source code of multiple languages separately. We take source code of an application and apply reverse engineering using Enterprise Architect Tool by creating an initial RDB model. The initial RDB model is processed by J2EE Pattern Detection Tool (JPDT) JPDT tool. JPDT uses JPSP module to parse multilingual source code and extend the initial RDB model created during initial parsing process.

JPSP is a super parser which contains support for source code parsing and searching using multiple search techniques i.e. using parsers, Regex, simple string search with custom rules. Initially JPSP contains four parsers and mappers for Java, JSPs, XML and XML based DLS, HTML, but the architecture of JPSP is easily extendable for adding support for a newer language by building a new parser or mapper.

Upon plugging in the initial RDB model created out of raw source code using Enterprise Architect, JPSP transforms the initial model into an extended super model. During this process, some of the old tables are enhanced to accumulate extended information. Moreover, some new tables are added, which improve the capability of the Initial-RDB Model. JPSP performs the following additions and upgradations.

a) Detection of Associations BY using JPSP Module:

Pattern detection by mining the interclass relationships is one of many techniques for design pattern detection. But detecting pattern instances from a toy dataset, may only needs a simple form of associations between two classes on the other hand in applied or industrial strength applications, the programmers use different strategies to apply associations (discussed in limitations of E.A Tool). These associations are necessary to identify interclass relationships like 'Delegation' and 'Uses' which are required for the detection of a J2EE pattern. For example, a DAO Pattern 'uses' Data Transfer Object or Value Object for the extraction and storage of data. This property reduces extensive remote database calls and network traffic. In-order to extract 'Uses' relationship from source code, we first need to extract all types of association that can statically be created in source code (E.A can identify only one type of association).

Extracting local variables and resolving their scope: Enterprise architect's parser could not detect associations, which are created by using a local variable of another class inside its function's body. Consider an example

```
Public Class a {
void function () { `
B   InstanceOF B = new B ();
}}
```

In this example, the class A has association with Class B. In-order to extract this type of association, we first need a full featured parser which can parse all local variables of a class, and then we need a 'Symbol Table' to resolve the scope of those local variables.

For this purpose, JPSP uses JavaCC parser with Java grammar and we have also used an abstract syntax tree of Java

Parser. This parser gives complete abstract syntax tree and number of visitors to traverse the abstract syntax tree. Java Parser generated by JavaCC can only detect all object creation statements defined in Java class. The generated parser and tree visitor has no mechanism to resolve a local variable and its scope.

Java Parser can parse any object creation statement like B InstanceOF B = new B (); but it can't actually resolve the scope e.g. what is 'B' and in which package and file 'B' is defined. In other words, we have to resolve the type of local variable.

Another aspect is that; detection of object creation statement does not describe that the object is a local variable or an attribute. Therefore, in-order to resolve the scope of a variable (local/Global), we need to know that whether the object is created inside a function or not. For this purpose, all classes and their functions are needed to be parsed from source code.

Fortunately, EA parser provides the information about classes in t_object table and operation names in t_operation table. Therefore, we need to parse all function types and then link our object declaration statements parsed with JavaParser in EA model by matching the type parsed with parser to the types of EA model.

In many cases, the Initial model created by EA parser do not provide a complete information about many of the aspects required from source code, however, much of the information can be gathered from different tables of EA model e.g. method name is present in t_operation, and method parameters are present in t_operationParms and parmeter's type is present in t_object table. We have extended t_operation table with another column "Operation_Signature" by extracting method signature and then inserting complete signature with a method name. In this way, it becomes a lot easier and faster to resolve local variables inside a function. Otherwise we have to repeatedly search function types at run time.

Using symbol table for resolving types: Resolution of type in an object creation is necessary to determine that a class A has association with class B. For this purpose, a symbol table is required. A symbol table is a structure which contains all classes names/references and their objects. Our symbol table not only captures the name of class but it also contains the function in which an object is created, Global variables/ attributes have empty function part. Fortunately, we do not need to parse class names, packages names or function names. EA has already built these metrics inside the initial model. A new table t_localVariable (Symbol Table) is created by JPSP that contains all object instances and then merge the object creation statements with respective operation_id and object_id. The operation_id refers function information in t_operation table and the object_id refers to a record inside the t_object table. Moreover, the t_localVariables table contains the reference to a class as an object creation type. The t_LocalVariable has a structure mentioned as:

Object_ID -:::- OperationID -:::- VarName -:::- VarType-:::- VarValue

It is important to mention that VarType is calculated by matching the package name of class creating object and then matching the varType with all class names available in the package of t_object table.

b) Addressing Weak Associations of EA Tool by JPSP Module:

In this section, multiple types of associations are addressed by using J2EE super parsing module.

- *Detecting Associations through Local Variable:*

In order to detect this type of associations, EA model is extended. The initial model created by EA contains a table t_connector, that stores different interclass relationships in the form of connector_Type, Start_Object_ID and end_Object_ID. We can compare local variables of each class by matching their types with classes in t_object (All Classes), if a match is found then this class is added as end_object of t_connector table as a new record (relationship). If the class type is not matched with the object, then it is assumed that the object is created by using some library e.g. its source code is not available.

- *Detection of Association through Operation Parameter*

Enterprise Architect is not capable to detect the association through parameters. Consider a scenario; an object of a 'Class A' has to use 'Class B' for a specific purpose e.g. 'Class B' is providing data to 'Class A' by grouping different elements into an object. Then instead of creating a class level attribute, the object of 'Class B' is created with in the function parameter of 'Class A'. Thus, we can say that the 'Class A' has association with 'Class B'.

In-order to determine this type of association, we need to query EA model because model already contains enough information about function names and their parameters and their types. Following information is already available.

- t_operation table contains attributes as function name.
- t_operationParams table contains parameter and their types.
- t_object table contains information including classes, interfaces, abstract classes, structs etc.

So, we only need to query EA model to resolve parameter types and then based on parameters type, we added a new record in t_connector table as a new association type. We use following queries to our model.

- Get All Classes from t_object table.
- For each class get all function parameters.
- For each classes function parameters, match class names from t_object table.
- Insert a new row as, Start_object_Id from source class and End_object_ID as matched class.

TABLE XIII. QUERY FOR FINDING ASSOCIATIONS IN FUNCTION PARAMETER

Function Parameter Association	"select distinct t_operation.Object_ID From t_operation cross join t_operationparams where t_operation.Object_ID =" + iStartObjectID + " AND t_operation.OperationID t_operationparams.OperationID" +" And t_operationparams.Type like '" + sEndObjectName + "'";

- *Detection of Association through Function Return Type:*

This type of association can also be extracted by using suitable queries to EA model. It is quite possible that a developer simply caste 'Class C' to 'Class B' and return from 'Class A' or uses 'functionA ()' to convert 'Class C' to 'Class B' and then return 'Class B' from 'Class A'. Thus, it is neither passing the 'Class B' through parameter nor it is creating an object of 'Class B', but it is creating an association of 'Class A' with 'Class B' through function return type. We can query in flowing manner to resolve function return type associations.

- Get all classes.
- Get each class's all function except function with void return type.
- Get each function's return type.
- Matches return type in class names.
- If match found, add source class as start_object_ID and matched class as end_object_ID in t_connector table as new function return type association.

TABLE XIV. QUERY FOR FINDING FUNCTION RETURN TYPE ASSOCIATION

Association with function return type	select Object_ID from t_operation Where Object_ID = "+iStartObjectID+ " AND t_operation.Type = '"+sEndObjName+"'"

c) Detection of Delegation Relationship

Delegation is a common relationship that exists in different components of a pattern. For Example, in Intercepting Filters, the Filter chain object must delegate the client's request to appropriate filter class. After applying the filter, the authenticity of the request is checked according to defined rules e.g. If a client wants to access the admin panel of a website, the intercepting filter is set to capture all the incoming requests and passes them to admin authentication filter. The role of the user is extracted from the given http request. The admin authentication filter verifies the user's role. If authentic user is found, the request will be sent to the requested services, otherwise an appropriate error message will be given back to the user.

Intercepting filter basically make it very simple to add preprocessing of all incoming requests to the server. Its structure makes it very simple to add or remove new filters, due to the decoupling between filter chain and filter object class. From reversing engineering perspective, the important point is the relationship between the 'filter Chain' object and between 'filter class'. Because of a diverse range of filter requirements, it is quite possible that the filter class appears just as a normal class, thus, the only way to extract out a filter class is to first detect a filter chain object and then find "Delegation Relationship" between all other available object, that is, Delegation relationship is necessary in pattern's reverse engineering definitions. Unfortunately, Enterprise Architect's initial model neither provides delegation information nor any support to extract delegation relationships from already available abstract source code.

- *Detecting Delegation by Call Scope Table:*

In-order to extract complex delegation relationship from source code, JPSP module use Java Parser. It is important to note that Java Parser do not parse delegation directly. Java Parser statically detect all "Calls" in a class by giving complete call statement and its scope name (name of the variable from where the call is initiated). For example

```
Class A
{
Class B b = new B ();
b.doSomething ();
}
```

Java Parser parse 'b.doSomething ()' statement.

There are two important key points for delegation detection.

- Delegation occurs only when one class calls the function of some other class to perform some task.
- Delegation does not take place when a function calls a local function (function of the same class).

In-order to detect, weather the function belongs to the same class or to some other class, we check three conditions.

- The calls using 'This' keyword are local calls and can't be considered in call scope table.
- The calls without any scope are also local calls and can't be included in call scope table.
- Resolution of scope's type is necessary e.g. in 'b.doSomething ()' we need to resolve the type of b by using 'Symbol Table'.

In-order to avoid runtime comparison, 'Call Scope' table also includes EndObjectId (the object from where the delegation is made). The endObjId is resolved by matching with 'scope name' e.g. 'b' from 'local variable' table and then extracting out the value of VarType and ObjectID. It is noticed that this matching (VarType and ObjectID) helps in successful mapping of endObjectId in 'CallScope' table with information presented in EA's model.

- *Detecting Delegation Relationships in J2EE Patterns:*

Design pattern mining techniques use interclass relationships which depend on inheritance association, composition and delegation etc. However, delegation has a key role for the detection of J2EE Patterns. Usually one source code artifact is involved in delegation with some other artifact. Upon the successful detection of delegation relation between source code artifacts leads to the detection of other components of a pattern and thus detecting the whole pattern instance. For example, 'Service locator Pattern' is invoked by 'Client' object. The client usually gives the name of service to be located from 'JND registry services'. Thus, client give name to 'service locator' which uses 'Initial context' object to resolve the type of service (The target object). The 'Initial Context' object is a component which can only be detected by inferring delegation relation between Service Locator object with other objects e.g. If SL has 'delegation' with any object by passing the object the name of service, then, that object is 'Initial context' object .

The JPSP extends Enterprise Architect by defining a new table t_delegations, which is basically a Call Scope table. Delegation relations are furnished at run time by checking the type of Call scope and then examining the function parameters list e.g. in InitialCtx.FindService("service"), type of Initial Context is resolved and then it is checked that weather this call is receiving some parameter or not. The checking of delegation process is not further ensured because of couple of reasons.

- In the presence of 'Call Scope Table', delegation checks are despicable due to the redundancy of information.
- Most of the times, single object delegate information to other objects by several separate calls. Resultantly a significance amount of information is required to be stored to treat each delegation separately. Whereas, only one clue of presence of delegation relation is necessarily required in the detection of a pattern component.

B. JPDT: J2EE Patterns Detection Tool

J2EE/JEA Pattern Detection Tool (JPDT) is a prototype tool that performs flexible static analysis of enterprise applications. This tool uses pattern definitions to analyze the input source and is capable to detect J2EE Patterns. Detected results can be helpful in overall analysis of J2EE Multilanguage application.

Prototype model of JPDT contains the following three modules.

1) *J2EE Pattern's Detection Engine (JPDE).*
2) *J2EE Pattern's Super Parser (JPSP).*
3) *J2EE Pattern's Visualization Module (JPVM).*

These Modules combined together form the parent project (an EA's plugin) to detect J2EE pattern within the source code of J2EE Enterprise applications.

1) J2EE Pattern Detection Engine (JPDE):

This module deals with the automatic recovery of J2EE Patterns from multilingual source code of J2EE applications. This module contains a repository for the definition of J2EE Patterns in the form of customizable feature types and pattern detection algorithm. The pattern detection algorithm use enhanced information of multilingual source code updated by super parser for the recognition of J2EE Patterns. JPDE has flexible and extendable nature to accommodate new patterns definitions or pattern's variants.

2) J2EE Pattern's Super Parser Module (JPSP).

JPDE is assisted by parser module (JPSP). This module addresses the limitations of Enterprise Architect Tool. During initial parsing some valuable information is missed by EA Tool. This information includes association through function parameters, local variables, function return types and delegation among artifacts. The Super Parser extracts this information and extends the existing model. This module has following features.

- Capable to parse General Purpose Languages, Web Based Languages and Domain Specific Languages (DSLs).
- Capable to detect delegations as well as weaker forms of association.
- Extract cross language associations from multilingual source code.
- Extendable to parse new languages.

3) J2EE Pattern's Visualization Module (JPVM).

The third component of J2EE Pattern Detection Tool is the visualization module that provides the metrics of recovered pattern's instances. This module offers navigational feature to move across various components and finding the source code dependencies. The navigational feature is capable to search both pattern's and non-pattern's components.

The visualization results are provided in the form of components (columns) and their instances (rows). The components are clickable instances of a pattern that can be navigated from source code. The user can precisely analyze the source code. Moreover, the cross language associations are placed in the same tabular format. This aspect helps in visualization of cross language dependency analysis. Our future plan is to extend the visualization module by providing the UML diagram of J2EE Pattern's instances.

C. J2EE Pattern's Detection Process:

The process of J2EE Pattern Detection (explained in Fig. 2) works in following order -

1) In the first step, Enterprise Architect (EA) recovers source code of multiple languages into its RDB model. The RDB model is an initial model that contains abstract information of parsed source code. This information is further used by J2EE Pattern Detection (JPDT) Tool.

2) In this step, the initially parsed source code is presented to the super parser. The super parser detects missing delegations and associations from multilingual source code.

3) In this step, the original source code along with reversed DB model and try to find missing aspects needed for Multilanguage source code analysis. The facts extracted by the super parser are mapped to RDB model.

4) In this step, patterns definitions are selected from pattern definitions catalog. The J2EE patterns are defined by using customizable and reusable feature types.

5) The J2EE Pattern Detection Tool (JPDT) contains analyzer that mines through the information from super RDB model and compare them with features of pattern's definitions.

6) In this step, detected patterns are represented by using prescribed metrics.

D. J2EE Pattern's Visualization Module (JPVM):

The recovered J2EE Pattern's instances are represented through J2EE Pattern's Visualization Module (JPVM) in the form of components that constitute a J2EE Pattern (Fig 3). This module provides source code navigation and access to particular pattern instance.

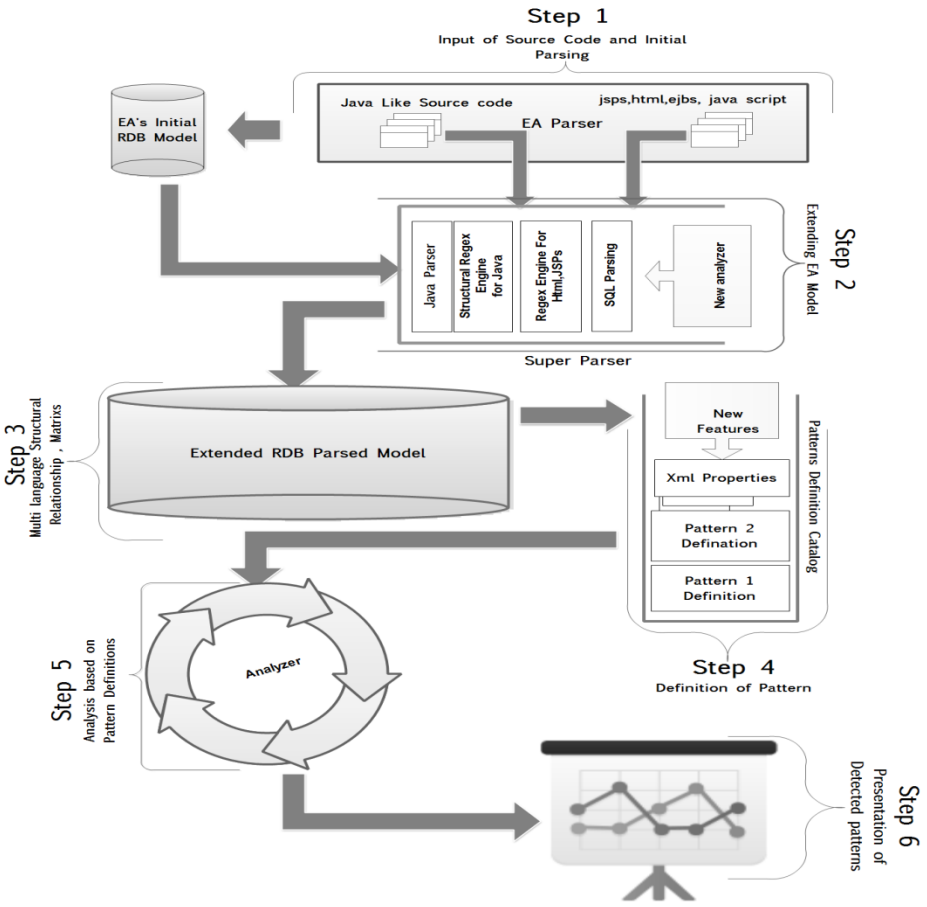

Fig. 2. J2EE Pattern Detection Tool

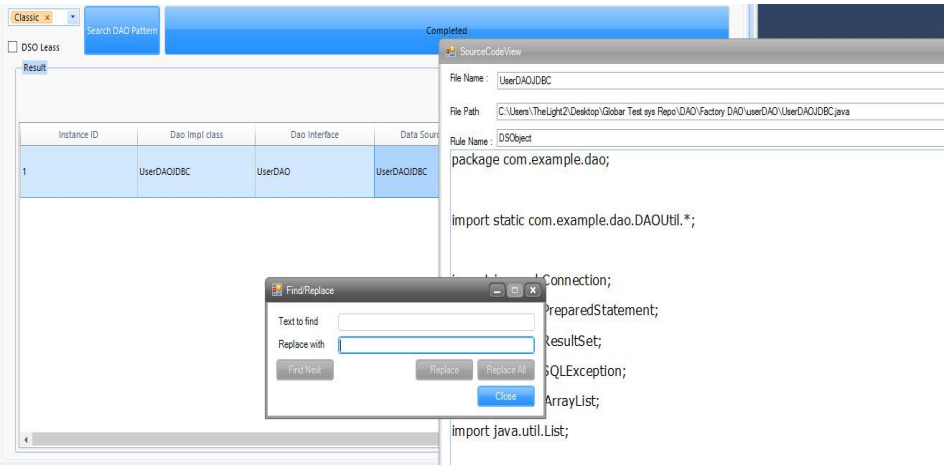

Fig. 3. Visualization of a Pattern through JPVM

VII. EVALUATION

This research lays a foundation for the detection of J2EE patterns. There is not approach available in the research to recognize J2EE Patterns from source code. Therefore, the performance of proposed system needs to be critically evaluated. In order to observe the capability of the prototype and completely recognize J2EE Patterns from enterprise applications, the pattern's definitions need to be comprehensive and precise and the detection algorithms is required to be perfect and effective.

A. Project Selection

The purpose of conducting this experimentation is to validate the quality of our approach in terms of completeness and effectiveness. Since detection of J2EE Patterns has not been presented before and we are setting a base line for future research, therefore, an extra care is required to evaluate our

approach. We ensure the following considerations to select the appropriate applications.

1) Open source J2EE applications i.e. their source code is available for evaluation;

2) The selected applications must be of applied nature and being used in the industry;

3) The applications contain maximum number of J2EE Patterns; and

4) The selected applications are of different sizes and complexity levels.

B. Determining Baseline

It is already mentioned that recognition of J2EE Patterns is presented for the first time; therefore we cannot compare the results with any previous approach. Another concern in validation process is the computation of correctness and completeness. The size and complexity in enterprise applications is quite high, therefore determining accuracy and entirety in J2EE applications is quite difficult and challenging. The correctness of proposed approach can be measured by comparing the recovered instances of J2EE Patterns with their definitions. However, the completeness of the approach is very difficult because there is not documentation available that provide details of J2EE Pattern's instances. Due to the large size of source code, the manual verification is not possible. Therefore, we are limited with the information shared with us. Tables 17-19 lists the metrics of the selected software applications.

C. Project Evaluation

The proposed approach is validated on two types of application environments.

1) Corpus of Test System's Repository (CTSR)

2) Open Source Enterprise Applications

1) Corpus of Test System's Repository

This Corpus contains the repository of more than 23 source code examples of J2EE Patterns from recognized resources [6, 8, 9, 15]. The main benefit of using this test system repository is that the manual validation is possible and number of available patterns is already known.

All pattern instances were manually validated and were found correct, this validate that our developed features were precise enough to extract these patterns from a large source code base. The statistics of results of test system repository are presented below.

2) Open Source Enterprise Applications

In order to evaluate our system, we choose different medium and large scale open source enterprise applications. All of these applications are well-known and are functional in the software industry. These applications are selected after thorough search, study of documentations and discussions with the software community, associated with the development of ERP applications. Source code of these applications is already available either on their corresponding websites, source forge or on GitHub. Table 16 provide name of open source applications along with their selected versions.

EJBCA or Enterprise Java Beans Certificate Authority [30], is an open source enterprise application which is based on Public Key Infrastructure (PKI) Certificate Authority (CA) [31-33]. It is a fully functional integrated certificate authority developed under J2EE technology. EJBCA provides complete PKI infrastructure in the form of large scale enterprise solution.

Openbravo [34], is an open source, web-based commerce and business ERP suit for small and medium-sized organizations [35-39]. Open bravo is developed under Java EE Platform that uses Contexts and Dependency Injection (CDI) and provides modern multi store management and retail ERP solutions.

Apache OFBiz [40], is an open source integrated suite under Apache Software Foundation. The OFBiz is a J2EE based ERP solution that contains framework components and business applications [41-43].

GeoServer [44], is an open source server for sharing geospatial data. It is a Java J2EE application that publish data from spatial data source using open standards [45, 46]. These services can be integrated with enterprise applications to create amazing mapping applications or integrate maps and GIS capabilities into existing web, mobile and desktop applications.

Java Pet Store [47], is a sample application, developed under Java Blue Prints program by Sun Microsystems. It is a reference application for Ajax web applications on Java Enterprise Edition Platform that uses J2EE Patterns.

TABLE XV. SELECTED OPEN SOURCE APPLICATION

Application	Version
EJBCA [30]	ejbca_ce_6_3_1_1
Geoserver [44]	geoserver-1.7.0
Ofbiz [40]	Apache OFBiz 16.11.01
Openbravo [34]	openbravo-3.0
Java Pet Store [47]	petstore-1_3_1_02

The source code information metrics are provided in Table 17. Most of the open source applications are medium and large enterprise level solutions. During the process of source code parsing and detection of J2EE Patterns, object oriented and cross language metrics were also recovered which are mentioned Tables 18 and 19 respectively.

Discussion

Our prototype tool recovered healthy number J2EE Pattern's instances from every application. The results of recovered J2EE Pattern's instances are provided in Table 20.

It is important to note that the selected applications are large and composed of thousands lines of code. Therefore, these applications are analyzed by keeping in mind that the manual validation the number of all applied patterns is not possible. The detection results are manually validated, by opening all the components in our pattern browser tool's and then manually inspecting the source code for further validation. Initially some false positives were found but after looking into definitions and further refining the definitions the entire false positive were disappeared. The metrics of the results show single instance for some of the J2EE patterns, because some

patterns have single or very few instances for the whole application.

The approach presented in this research, has shown 100 % results when evaluated on repository of source code examples of J2EE patterns. In analyzing open source enterprise applications, we also found promising results. However, few J2EE Pattern's instances were not recognized. We investigated this problem by manual inspection of these pattern instances, it is found that these patterns were implemented in the source code but their implementation is deviated from the original definitions provided by sun micro system. They were present with a variation and do not qualify for the actual definition of J2EE Patterns. The variation detection of J2EE Patterns is another aspect of design pattern research. We have tried to accommodate more and more Pattern definitions but do not attempted to deviate from the core definitions of J2EE Patterns.

TABLE XVI. METRICS FOR SELECTED OPEN SOURCE APPLICATIONS

Source Code Metrics		CTSR (Corpus)	Open Source Enterprise Applications				
			EJBCA [30]	GeoServer [44]	OFBiz [40]	Openbravo [34]	Java Pet Store [47]
Size	Application size	45.7 MB	57.4 MB	104 MB	146 MB	380 MB	11.1 MB
Directories	Directories	2,882	980	1,040	1,745	1,591	378
LOC	Lines of Code	238,152	357,952	192,403	356,474	434,043	6,573
BLOC	Blank Lines of Code	33,164	39,871	28,745	39,221	44,596	4,603
SLOC-P	Physical Executable Lines of Code	98,104	230,877	98,738	259,761	306,605	17,891
SLOC-L	Logical Executable Lines of Code	67,147	174,124	74,019	203,697	221,021	13,957
MVG	McCabe VG Complexity	10,051	23,501	13,867	43,723	38,267	1,796
C&SLOC	Code and Comment Lines of Code	1,329	2,241	571	771	2,109	77
CLOC	Comment Only Lines of Code	114,215	87,204	64,920	57,492	82,842	14,079
CWORD	Commentary Words	603,781	505,004	276,208	392,418	508,444	103,222
HCLOC	Header Comment Lines of Code	48,561	20,230	3,778	20,805	32,930	10,828
HCWORD	Header Commentary Words	156	122,211	26,577	149,924	240,627	86,048

TABLE XVII. OBJECT ORIENTED METRICS PARTICIPATED IN J2EE PATTERN DETECTION PROCESS

Metrics	CTSR (Corpus)	Open Source Enterprise Applications				
		EJBCA [30]	GeoServer [44]	OFBiz [40]	Openbravo [34]	Java Pet Store [47]
Packages	227	614	144	276	198	128
Total classes	1,184	2,121	1,121	1,135	1,987	267
Abstract Classes	59	181	64	100	83	21
Interfaces	104	212	76	90	68	63
Methods	12,274	36,446	9,885	15,544	14,818	1,955
Attributes	3,714	13,253	3,275	6,153	7,232	1,132
Associations	713	158,334	4,826	15,185	21,662	4,307
Generalizations	504	1,225	557	707	1,318	43
Realizations	185	439	134	263	227	29
Total Connections	1,061	166,742	5,624	22,221	23,362	4,385

TABLE XVIII. CROSS LANGUAGE METRICS PARTICIPATED IN J2EE PATTERN'S DETECTION PROCESS

Cross Language Metrics	CTSR (Corpus)	Open Source Applications				
		EJBCA [30]	GeoServer [44]	OFBiz [40]	Openbravo [34]	Java Pet Store [47]
Java Files	1705	3,823	1,413	2,139	2,387	467
XML Files	252	3252	405	2,732	2,341	97
HTML Files	300	554	75	46	450	37
JSP Files	394	125	146	140	1	98
SQL Files	25	29	5	11	122	5
All Parsed Files	2358	6168	1,669	4,076	4,746	541
Other Language Files	1173	3,418	3,064	5,813	5,753	206
Total Files	3531	9,586	4,733	9,889	10,499	747
Cross Lang Associations	23730	141638	2,199	2,787	18,862	3,729

TABLE XIX. J2EE PATTERN'S INSTANCES RECOVERED FROM OPEN SOURCE ENTERPRISE APPLICATIONS

J2EE Patterns	CTSR (Corpus)	Open Source Applications				
		EJBCA [30]	GeoServer [44]	OFBiz [40]	Openbravo [34]	Java Pet Store [47]
Composite View	1	2	13	11	11	1
Front Controller	2	2	1	-	-	3
Intercepting Filter	3	4	-	1	2	1
Business delegate	2	-	1	-	-	3
Session Façade	4	-	-	-	-	10
Value List handler	2	1	49	41	1	-
Service Locator	9	1	-	-	-	5
Value Object	10	25	21	4	-	-
Data Transfer Object	21	207	49	21	23	3
Data Access Object	14	1	2	-	5	2
Total J2EE Pattern's Instances	68	243	136	78	42	28

Threats to Validity:

In this section, threats for the acceptability of subject system are discussed. In order to deal with external validity, we need to ensure that the presented approach is generalized and scalable for the large systems. For this purpose, the proposed approach is evaluated on two types of system 1st corpus of test system's repository from reliable resources and 2nd analysis of open source enterprise applications that are already implemented and their documentation is available. The corpus of applications contains small, medium and large source code. The extracted patterns are manually verified from the source code, the results support our approach.

This research is of unique nature that lay foundation for the recognition of J2EE Patterns from all tiers of JEE Platform. The threat foreseen for internal validity is the standardizations of J2EE pattern's definition. For this purpose, we extracted the properties from core definitions of patterns from sun micro systems and other reliable resources. We translated these properties into extendable and customizable feature types. The J2EE patterns are re-defined on the basis of feature types. The proposed J2EE design pattern detection tool (JPDT) used these definitions and successfully recognized J2EE patterns from source code.

VIII. CONCLUSION AND FUTURE WORK

Java enterprise applications (JEAs) support development of flexible and lightweight distributed applications. These applications are composed of multilingual source code artifacts. J2EE Patterns helps to build effective enterprise applications. This research involves development of semi specification and feature types of J2EE Patterns of enterprise application that leads to the recognition of J2EE Patterns from open source enterprise applications. At first, properties of J2EE patterns are extracted from reputable resources of Java enterprise patterns. These properties are then converted to definitions in the form of semi specifications and feature types. A pattern detection criterion is built on the basis of these semi specifications and feature types. These features are extendable which can be translated in the form of multiple techniques for the recognition J2EE Patterns and further analysis of multilingual applications. Second, process for the recognition of J2EE Patterns is presented by extracting the Feature Types. This process is implemented as J2EE Pattern Detection Tool (JPDT), which contains the definitions of J2EE Patterns. JPDT enhances the capability of enterprise architect tool and recovers JEA/ patterns form source code of enterprise applications. This tool is evaluated on test repository and on open source java enterprise applications. Multilingual source code analysis by extraction of multi-language artifacts is another aspect and is future prospect of this research.

REFERENCES

[1] P. Benedusi, A. Cimitile, and U. De Carlini, "Reverse engineering processes, design document production, and structure charts," Journal of Systems and Software, vol. 19, pp. 225-245, 1992.

[2] J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," IEEE software, vol. 7, pp. 13-17, 1990.

[3] G. Rasool and P. Mäder, "A customizable approach to design patterns recognition based on feature types," Arabian Journal for Science and Engineering, vol. 39, pp. 8851-8873, 2014.

[4] Z. Mushtaq and G. Rasool, "Multilingual source code analysis: State of the art and challenges," in 2015 International Conference on Open Source Systems & Technologies (ICOSST), 2015, pp. 170-175.

[5] Bow-Wow, Pet Architecture Guide Book: World photo Press, 2001.

[6] Alur, D. Malks, J. Crupi, G. Booch, and M. Fowler, Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies: Sun Microsystems, Inc., 2003.

[7] N. Shi and R. A. Olsson, "Reverse engineering of design patterns from java source code," in 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), 2006, pp. 123-134.

[8] J. Crupi and F. Baerveldt, "Implementing Sun Microsystems' Core J2EE Patterns," Compuware White Paper, 2004.

[9] D. Alur, J. Crupi, and D. Malks, "Core J2EE Patterns 2nd," ed: Prentice Hall, 2003.

[10] W. Crawford and J. Kaplan, J2EE design patterns: " O'Reilly Media, Inc.", 2003.

[11] Flores, A. Barrón-Cedeno, L. Moreno, and P. Rosso, "Cross-Language Source Code Re-Use Detection Using Latent Semantic Analysis," Journal of Universal Computer Science, vol. 21, pp. 1708-1725, 2015.

[12] K. Cemus, T. Cerny, L. Matl, and M. J. Donahoo, "Aspect, Rich, and Anemic Domain Models in Enterprise Information Systems," in International Conference on Current Trends in Theory and Practice of Informatics, 2016, pp. 445-456.

[13] Stearns, S. Brydon, I. Singh, T. Violleau, V. Ramachandran, and G. Murray, Custom Edition of Designing Web Services with the J2EE™ 1. 4 Platform, JAX-RPC, SOAP, and XML Technologies: Addison-Wesley, 2004.

[14] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," IEEE transactions on software engineering, vol. 32, pp. 896-909, 2006.

[15] Deepak, J. Crupi, and D. Malks, "Core J2EE patterns," Rio de Janeiro: Campus, 2002.

[16] R. Johnson and J. Hoeller, Expert one-on-one J2EE development without EJB: John Wiley & Sons, 2004.

[17] J. Crupi, D. Malks, and D. ALUR, Core J2EE Patterns: Gulf Professional Publishing, 2001.

[18] Y. Coppel and G. Candea, "Deprogramming Large Software Systems," in HotDep, 2008.

[19] Costagliola, A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, "Design pattern recovery by visual language parsing," in Ninth European Conference on Software Maintenance and Reengineering, 2005, pp. 102-111.

[20] J. Dong, D. S. Lad, and Y. Zhao, "DP-Miner: Design pattern discovery using matrix," in 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), 2007, pp. 371-380.

[21] O. Kaczor, Y.-G. Guéhéneuc, and S. Hamel, "Efficient identification of design patterns with bit-vector algorithm," in Conference on Software Maintenance and Reengineering (CSMR'06), 2006, pp. 10 pp.-184.

[22] M. Zanoni, F. A. Fontana, and F. Stella, "On applying machine learning techniques for design pattern detection," Journal of Systems and Software, vol. 103, pp. 102-117, 2015.

[23] J. K. van Dam, "Identifying source code programming languages through natural language processing," 2016.

[24] Rasool and P. Mäder, "Flexible design pattern detection based on feature types," in Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on, 2011, pp. 243-252.

[25] K. Deeptimahanti and M. A. Babar, "An automated tool for generating UML models from natural language requirements," in Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 680-682.

[26] P. Khodabandehloo and H. L. Reed, "Design tool and methodology for enterprise software applications," ed: Google Patents, 2010.

[27] F. Matthes, S. Buckl, J. Leitel, and C. M. Schweda, Enterprise architecture management tool survey 2008: Techn. Univ. München, 2008.

[28] G. Sparks, "The business process model," Enterprise Architect, Wien, pp. 1-9, 2000.

[29] G. Sparks, "Enterprise architect user guide," 2009.

[30] P. S. AB. (2016, 01102016). EJBCA Enterprise Available: https://www.primekey.se/technologies/products-overview/ejbca-enterprise/, https://www.ejbca.org/index.html

[31] N. A. Devi and M. Sundarambal, "Secured Web Service Communication using Attribute based Encryption and Outsource Decryption with Trusted Certificate Authorities (ABE-TCA)," Asian Journal of Research in Social Sciences and Humanities, vol. 6, pp. 896-911, 2016.

[32] S.-Y. Tan, W.-C. Yau, and B.-H. Lim, "An implementation of enhanced public key infrastructure," Multimedia Tools and Applications, vol. 74, pp. 6481-6495, 2015.

[33] Bruneo, F. Longo, G. Merlino, N. Peditto, C. Romeo, F. Verboso, et al., "A Modular Approach to Collaborative Development in an OpenStack Testbed," in Network Cloud Computing and Applications (NCCA), 2015 IEEE Fourth Symposium on, 2015, pp. 7-14.

[34] S. L. U. Openbravo. (2016, 01102016). Openbravo. Available: http://www.openbravo.com/product-download/

[35] L. Coppolino, S. D'Antonio, C. Massei, and L. Romano, "Efficient Supply Chain Management via Federation-Based Integration of Legacy ERP Systems," in International Conference on Intelligent Software Methodologies, Tools, and Techniques, 2015, pp. 378-387.

[36] K. B. C. Saxena, S. J. Deodhar, and M. Ruohonen, "Organizational Practices for Hybrid Business Models," in Business Model Innovation in Software Product Industry, ed: Springer, 2017, pp. 95-107.

[37] S. Bajaj and S. Ojha, "Comparative analysis of open source ERP softwares for small and medium enterprises," in Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on, 2016, pp. 1047-1050.

[38] M. Bahssas, A. M. AlBar, and M. Hoque, "Enterprise Resource Planning (ERP) Systems: Design, Trends and Deployment," The International Technology Management Review, vol. 5, pp. 72-81, 2015.

[39] Johansson and F. Sudzina, "ERP systems and open source: an initial review and some implications for SMEs," Journal of Enterprise Information Management, vol. 21, pp. 649-658, 2008.

[40] OFBiz®. (2016, 01102016). Apache OFBiz 16.11.01. Available: http://ofbiz.apache.org/download.html;

[41] R. Hoffman, Apache OFBiz Cookbook: Packt Publishing Ltd, 2010.

[42] M. Ellison, R. Calinescu, and R. F. Paige, "Towards Platform Independent Database Modelling in Enterprise Systems," in Federation of International Conferences on Software Technologies: Applications and Foundations, 2016, pp. 42-50.

[43] Wong and R. Howell, Apache OFBiz Development: The Beginner's Tutorial: Packt Publishing Ltd, 2008.

[44] O. S. G. Foundation. (2016, 01102016). GeoServer Available: http://geoserver.org/download/

[45] Xia, X. Xie, and Y. Xu, "Web GIS server solutions using open-source software," in Open-source Software for Scientific Computation (OSSC), 2009 IEEE International Workshop on, 2009, pp. 135-138.

[46] L. Sun, D. He, and P. Zhao, "A Research of Publishing Map Technique Based on Geoserver," Asian Journal of Applied Sciences, vol. 8, pp. 185-195, 2015.

[47] Chen, C. P. Ho, R. Osman, P. G. Harrison, and W. J. Knottenbelt, "Understanding, modelling, and improving the performance of web applications in multicore virtualised environments," in Proceedings of the 5th ACM/SPEC international conference on Performance engineering, 2014, pp. 197-207.