OPEN ACCESS
International Journal of Innovations in Science & Technology
IJIST
P ISSN : 2618-1630
E ISSN : 2909-6130

Realization of Presentation layer information of Legacy Java Enterprise Applications Through Design Pattern's Recovery

Zaigham Mushtaq[1], Ghulam Rasool[2]
[1]The Islamia University, Bahawalpur, Pakistan
[2]COMSATS University Islamabad, Lahore Campus.
Corresponding author: ZaighamMushtaq (zaigham@iub.edu.pk)
Citation | Mushtaq. Z and Rasool. G, Realization of Presentation layer information of Legacy Java Enterprise Applications Through Design Pattern's Recovery, International Journal of Innovations in Science and Technology. Vol 4, Issue 1, pp: 32-50, 2022
Received | Dec 20, 2021; Revised | Jan 21, 2022 Accepted | Jan 24, 2022; Published | Jan 26, 2022.

Abstract

The presentation layer is the outermost layer of an application that provides user interface and communication services. This layer is responsible for session management, controlling client access, and validations within data from the client. In legacy enterprise applications like Java Enterprise Edition Platform (Java EE), the design considerations of the presentation layer are spread over different design patterns and cross-language constructs. Resultantly, the analysis of such applications becomes quite challenging due to their heterogeneity, essentially required for the extraction of design-level information and further modernization. In this research, a flexible technique is presented to extract presentation tier information based on customizable feature types by recovering instances of presentation tier patterns of the Java Enterprise Edition Platform. The proposed approach is evaluated on well-operative open-source Enterprise Applications. The validation results demonstrate the extraction of presentation tier information through Design Pattern's recovery. This prototype is validated on the repository of source code of Java applications as well on open source java applications.

Keywords: Source Code Analysis, Design Patterns, Java Enterprise Appications,

1. Introduction

Evolution, bugs fixing and up-gradation are common in any software system. Many of these features result in the enhancement and customization of a software application's structural design. During the development and maintenance of software applications, consistency of documentation with the design of an application is essential. Legacy software systems [1-3] are difficult to maintain and upgrade due to obsolete or missing design documentation. It is also observed that the available documentation does not match the original design due to changes and enhancements made over time. The code comments and other sources may give some hints to the software developer to complete the objectives but that does not ignore the necessity of the complete architecture information of the system. Therefore, for the maintenance and up-gradation of the legacy software system, the software developers must be able to see and understand the complete architecture of the system to make modifications and apply best design practices [4].

Design patterns are recurring problem-solving techniques[5]. They are reusable components that can be utilized to solve certain design issues [8]. They aid in the improvement of quality of software system design [6]. In particular, the detection of different design patterns can help a great deal to understand the design decisions which can be useful for the comprehensive examination of a system.

The recovery of different design patterns can be very valuable and can help a great deal in software reverse engineering, maintenance, program comprehension, source code analysis, redesign, and re-engineering of software applications [7, 8]. Modification of a software system without a thorough knowledge of multiple design patterns, on the other hand, can cause the application logic and justification behind the implemented design pattern variation to change. Incomplete knowledge of design patterns can also make other aspects of software engineering more difficult, such as refactoring, restructuring, and technology upgrades. All forms of design patterns must be retrieved to gain insight into the system.

Software application heterogeneity has increased as a result of modernization, making applications more complex and analysis more difficult [8]. As previously stated, design pattern recovery is critical for extracting design-level information and the software application's intent.Such apps' design knowledge and internal logic are stored in many levels that are accessible. Information is dispersed across different tiers and languages in Java Enterprise applications, which features a layered architecture. The design artifacts are organized into separate components that reside on the computer. The presentation tier is the first layer and is responsible for handling user interfaces and bears communication logic. This layer encapsulates graphical design and user interaction code. The recovery of presentation tier logic [1] with the help of different design patterns can help a great deal for understanding and redesigning the structure of an application.

Recovery of different design patterns can help improve the reusability and extendibility of written logic. Different types of design pattern recovery approaches are reported that support the extraction of design-level information from software applications [9]. However, complete detection of presentation tier J2EE Pattern has not been presented so far. The existing approach supports the only partial recovery of J2EE Patterns within the presentation tier [1]. Therefore, to be able to see the presentation tier logic implemented, all the design patterns present in a system must be detected and visualized.

In this paper extendable approach is presented that supports the extraction of presentation information from the Java Enterprise Edition platform by recovering presentation tier design patterns. The proposed approach is extendable to support detection of other patterns like GOF Patterns etc.

The Section 2 in this paper describes related work by describing source code analysis and design pattern recovery. Section 3 presents background about the role of Enterprise Applications. Section 4 mentions the mechanism for the extraction of presentation Tier information by using design pattern recovery and Section 5 describes the conclusion and future work.

Background: Role of Design Patterns in Enterprise Applications.

Enterprise applications are large-scale, distributed, multilingual applications constructed with a variety of technologies. These apps' modules comprises of several language artifacts. Multilingual enterprise applications are best exemplified by Java enterprise applications. Enterprise applications enable the creation of numerous components utilizing programming languages such as C#, Java, HTML, JavaScript, SQL, DSLs, and XML. Because the information that needs to be fetched is distributed across numerous modules constructed using different

programming and scripting languages, the analysis of such an application is a tough and time-consuming operation. Enterprise applications are complicated and are composed of layers or tiers, each of which are composed of various technologies and has its own set of responsibilities. Furthermore, each layer has a collection of different sorts of design patterns to formalize the solution to the difficult problem [24, 41, 90]. Enterprise applications' server-side architecture is organized into three layers: The presentation layer, the Business Logic Layer, and the Data Access layer [6].

The enterprise applications are built using design patterns of many types like GOF [6]. Also, Java Enterprise application design patterns (JEA) [2, 3] are proven solutions that can deal with the complexity of enterprise-level applications by offering encapsulation. Patterns of Enterprise Application [5] are another type of pattern that is widely used in enterprise applications to implement and reuse complex logic. These different types are mostly used in enterprise-level applications. The reverse engineering of such various design patterns can help recover design information, architecture, and logic used in the application. Hence the pattern recovery techniques explained in the previous section can be applied with the help of static and dynamic analysis.

The high-level model of Enterprise Application is presented in Figure 1. The model explains that all tiers are formed using different components and each layer has its unique responsibility. The presentation layer encapsulate login to service a client request. The client's request is captured by the presentation layer, which then conducts the relevant procedures [3]. Single sign-on, session management, access control to business services, response construction, and response delivery to clients are all part of this operation.

The information is stored in the form of design patterns in presentation layers [10]. This information includes pre-processing and post-processing of a request. This layer has centralized control for handling requests. It also contains a protocol-independent object [11] to pass to other components. This layer handles view and action management. It creates, dispatches views, and handles login for view management.

Furthermore, the J2EE design patterns of the presentation tier are listed in Table 1. These patterns are verified and tested solutions that help build scalable enterprise applications. These patterns can be applied to any environment other than Java enterprise applications. In a nutshell, enterprise architecture follows tiered architecture, and each design pattern is specific to a layer. To extract design logic from the presentation tier, there is a need to detect different types of design patterns. Table 1. shows the presentation tier logic needed to be extracted for reverse engineering and the respective design patterns that lies in that information.

Table 1. Presentation Tier Design Patterns [2, 3]

Tier Name	Technology	Name of Patterns
The presentation tier	Applets, Servlets, UI Elements, Browser, JPS etc.	Dispatch View, Intercepting Filter, View Helper, Service-To-Worker, Front Controller, Composite View, Application Controller, Context Object

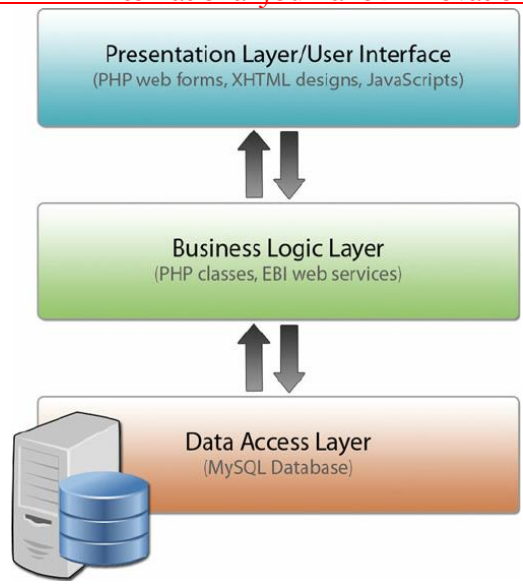

Figure 1. Tiered model of Enterprise Application [2-4]

Table 1. Enterprise Application Presentation Tier logic and relevant Design Patterns [3] [5]

Presentation Tier Information Concern	Name of Design Patterns
Session management	Service-To-Worker, Application Controller
Client Access Control	Front Controller, Dispatch View
Validations and Token Synchronization	Intercepting Filter
Helper Properties Integrity and Consistency	View Helper, Composite View
Protocol independent information	Context Object
Disparate Logic Localization	View Helper
Control Code in Multiple Views	Front Controller, Application Controller

Session management, Client Access Control, Helper Properties, Protocol independent information, Disparate Logic Localization and Control Code in Multiple Views become unavailable. Key characteristics of the discussion are inscribed in moving forward to the detection of various types of design patterns from enterprise applications are described below.

1. Each design pattern in an enterprise application has a unique requirement, which can help understand the reason for the implemented solution.
2. Detection of different design patterns can support reusability which can help maintain a simpler task with fewer resources to spend [3].
3. The presence of Enterprise applications makes re-engineering a necessary requirenment, as enterprise applications are found everywhere. The recovery of enterprise-level design patterns increases the adoptability rate and reusability.
4. The discovery of multiple design patterns that incorporate systems increases the reusability of diverse components and reduces cost, maintainability, and design consistency.

5. Different design patterns are applied to build cross-language enterprise applications, and they are heterogeneous. The recovery of such patterns is a technique for the analysis of enterprise applications.
6. In a tiered model, the information flows in a layer in a specific sequence from one component to the other. Therefore, to realize the complete information, all the patterns that participate in the presentation tier. However, to the best of our knowledge, available techniques [1-3, 10, 12] don't completely realize the presentation tier information using design pattern recovery.

Proposed Methodology

Only a few patterns from the presentation tier have been discovered, as previously stated [1-3, 10, 12], including Front Controller, Composite View, and Intercepting Filter patterns. Resultantly, valuable information about the presentation tier and its logic is lost inclusive of Session management, client verification, token synchronization, Integrity, and Consistency, etc. [2, 3]. Therefore, we cannot analyze the application properly which is a prerequisite for reusability, refactoring, reverse engineering, and re-engineering [3][13].

This research enhances the existing approach [1-3, 10, 12] by allowing the recognition of remaining presentation tier patterns counting Dispatch View, View Helper, Service-To-Worker, Application Controller, and Context Object patterns along with the already detected patterns [2, 12, 13].

At first, the catalog of feature types [1-3, 10, 12] for presentation tier patterns is enhanced, some of the additional features are added and definitions of remaining presentation tier patterns are taken into account using customizable feature types. Based on these definitions, the pattern detection algorithm is refurbished.

Features for Presentation Tier Design Patterns. A pattern definition's building blocks are features. The components and their interrelation are described by features. A design pattern is a grouping of several characteristics. In this section, the features for the detection of Presentation Tier Design Patterns [12] (Dispatch View, View Helper, and Service-To-Worker, Application Controller, and Context Object patterns) are presented.

Context Object

This pattern provides context-oriented access and is responsible for state encapsulation in a protocol-independent way, shared throughout the application. This modeled couples services and components and exposes only protocol-specific and context-based relevant APIs [12] for use.

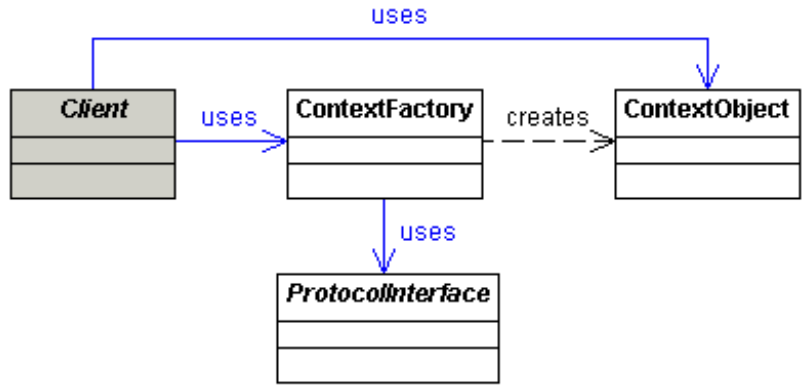

Figure 1. Context Object

Table 2. Features of Context Object Pattern

Index	F. #	Feature's Signature
PF1	F28	getAllClasses()

PF2	F45	Hasclass (PF1) Extends HttpServlet
PF3	F46	HasObject of HttpServletRequest
PF4	F47	HasObject of HttpServletReponse
PF5	F14	HasMethodWithRType (PF3, PF4)
PF6	F46 & F14	HasObject (PF3)AND HasMethodWithParameterType (PF3, PF4)
PF7	F14	HasMethodWithParameterType (PF3, PF4)
PF8	F14	HasMethodWithParameterType (PF6)
PF9	F19	HasRealization(PF7, PF8)

View Helper

This figure is used to resolve the complexity and streamline access to model state and data access logic. Sometimes business data access logic and presentation logic are intermingled. Resultantly, the reusability, flexibility, and change management become quite difficult. The view helper pattern supports template-based views and disallows the use of program logic in views. The panoramas are used to provide encapsulation of formatting code by delegating its responsibilities, whereas, Helper is utilized in encapsulation of view processing logic [12]. It acts as an adapter to process formatting logic.

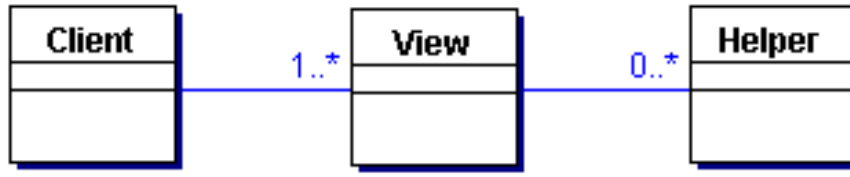

Figure 2. View Helper Pattern

Table 3. Features of View Helper Pattern

PF1	F29	GetXMlObjects ()
PF2	F30	HasNumberOfAssociationsWithType (PF1,>=2, “HTML” | “JSP”)
PF3	F31	HasTheseXMLTags (PF2, “Include”| “Put”)
PF4	F32	GetJSPObjects ()
PF5	F33	GetHTMLObjects ()
PF6	F30	HasNumberOfAssociationsWithType (PF1, >=1, “HTML” | “JSP”)
PF7	F5	HasAssociation (PF5, PF3)
PF8	F34	HasNoNumberOfAssociationsWithType (PF4 >= X, ”HTML” | ”JSP”)
PF9	F5	HasAssociation (PF7, PF3)

Dispatcher View

Dispatcher View invokes view processing before initiating the business process. This design is implemented with the help of the dispatcher component as the combination of Front Controller and View Helper patterns. The role of a dispatcher is to perform navigation or view management inside a controller or in view.

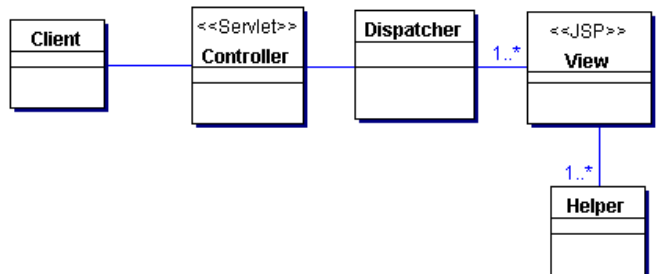

Figure 3. Dispatcher View Pattern

Table 4. Features of Dispatch View Design Pattern

PF1	F20	HasDefinedAType (AllObjs, "Dispatch")
PF2	F40	HasNoRealizationWithType (PF1, "HttpServlet")
PF3	F31	HasTheseXMLTags (PF2, "Include"| "Put")
PF3	F32	GetJSPObjects ()
PF4	F33	GetHTMLObjects ()
PF1	F5	HasAssociation (F5, F3)
PF2	F34	HasNoNumberOfAssociationsWithType (PF4 >= X, "HTML" | "JSP")
PF3	F5	HasAssociation (PF5, PF3)

Service-To-Worker

This Pattern performs authorization and authentication, encapsulates business logic, and simplifies control flows and views. The Service to Worker is a combined form of micro patterns including dispatcher or controller including helper or views. This pattern supports centralized control and request handling [12]. After that forwards control to view for presentation in the form of dynamic response.

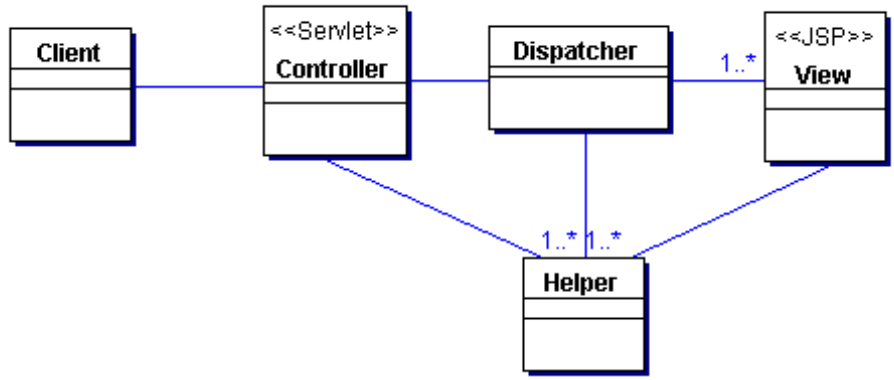

Figure 4. Service to Worker Pattern

Table 5. Service to Worker Pattern

PF1	F12	GetAllInterfaces ()
PF2	F5	HasAssociation (AllObjs, F1)
PF3	F15	HasMethodWithParameterType (AllObjs, F2| "Object"| "String")
PF4	F14	HasMethodWithRType (F3, F2| "Object"| "String"| "T")
PF5	F28	GetAllClasses ()
PF6	F15	HasMethodWithParameterType (F6,"String"| "string")
PF7	F41	HasNoDelegation (F4, F2)
PF8	F23	HasDelegation (F8, F5)
PF9	F19	HasRealization (AllObjs, F2)
PF10	F23	HasDelegation (AllObjs, F9)

Application Controller

This convention provides centralized retrieval and invocation components for request-processing (like commands and views) and offers a central point for screen navigation and application flow. This structure offers centralized and modularized actions and views management [12].

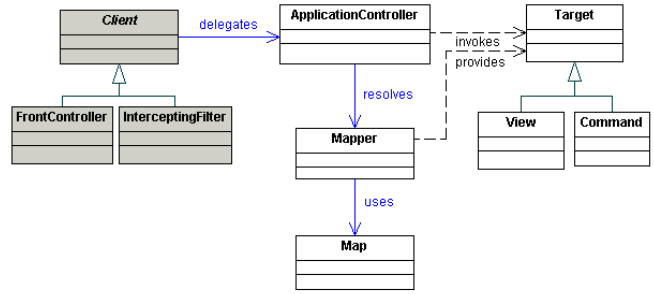

Figure 5. Application Controller Pattern

Table 6. Application Controller Pattern

PF1	F29	GetXMlObjects ()
PF2	F32	GetJSPObjects ()
PF3	F33	GetHTMLObjects ()
PF4	F30	HasNumberOfAssociationsWithType (PF3, >=1, “HTML” | “JSP”)
PF5	F5	HasAssociation (PF4, PF3)
PF6	F34	HasNoNumberOfAssociationsWithType (PF5 >=1, “HTML” | “JSP”)
PF8	F6	HasDTOs ()
PF9	F28	GetAllClasses ()
PF10	F5	HasAssociation (PF8, PF9)
PF11	F30	HasNumberOfAssociationsWithType(1, (“Class”&&”Interface”),PF4)

Extended Catalogue of J2EE Design Patterns by using Feature Types

The design Pattern is necessary for the production and detection of the pattern since it includes concrete definitions and standard parameters. As a result, the Presentation tier J2EE Patterns definitions are extracted from standard resources[1, 2, 13].

The features type for their realization is decided based on these definitions. The feature types, as previously said, are expandable and reusable, and can be translated into a pattern detection technique. These features can be developed to increase the quality of the image and find other patterns. Previously, the catalog of feature types of the J2EE Design Pattern was presented [1, 10]. However, only four patterns were realized to represent Presentation Tier Information including Front Controller, Intercepting Filter, and Composite View Patterns [1] . As a result, vital information about Presentation Tier along with Session management, Client Access Control, Helper Properties, Protocol independent information, Disparate Logic Localization, and Control Code in Multiple Views become unavailable.

In this research, all remaining Patterns of the J2EE Platform were realized to extract complete information of J2EE Patterns relating to the Presentation Tier. As the Pattern definitions are customizable & extendable based on feature types to accommodate new pattern definitions or their variants. The catalog of J2EE Patterns is further extended to accommodate new pattern definitions that pertain to the Presentation tier as well as Context Object Pattern, Application Controller Pattern, View Helper Pattern, Dispatch View Pattern, and Service to Worker Pattern.All the pattern definitions are developed by the existing catalog of feature types of J2EE Design Pattern[1, 10]. However, to cater to Servlet information three more features are introduced and added to the Catalogue of Feature Types.

Table 7. Extended Features of Features Catalogue

F. #	Feature Signatures

F44	Hasclass () Extends HttpServlet
F45	HasObject of HttpServletRequest
F46	HasObject of HttpServletReponse

Explanation of new Features

The subject class is an HttpServlet class that extends the generic Servlet Class. We can get specified methods of Servlet Class. Feature # 44 is for the class that returns Features of HttpServelet (mentioned in Table *8*).The role of HttpServletRequest, HttpServletResponse is to get and set HttepServlet methods. ServletRequest provides basic setter and getter methods for requesting a Servlet. HttpServletRequest extends the Interface with getters for HTTP communication. HttpServletResponse object receives the request from the service method and dispatches the request to the concerning method depending on the incoming HTTP request type. Feature # 45(Table 8) pertains to the object to receive incoming HTTP request headers and form data. Feature # 46(Table 8) pertains to the object to setup HTTP response including content type and response message.

Extended J2EE Pattern Detection Approach

The proposed approach is translated in the form of a Design Pattern Detection Tool that contains the definitions of Presentation Tier Patterns. This approach used the meta-model of the enhanced RDB model and realized the J2EE pattern instances from the source code.

The pattern detection approach for the presentation tier contains the algorithms that used the feature type of J2EE Patterns. A combination of feature types isemployed to extract accurate, interclass relationships among the design pattern components. The proposed approach identifies the features within the source code by using multiple kinds of classes, object-oriented relationships, interclass relationships, relevant objects, and a varity of methods. All these steps support the effective realizations of pattern instances within the Presentation tier.

To detect the complete catalog of J2EE Patterns for the Presentation Tier, the existing approach [1] needs enhancement as tool availability is deficient to observe all 9 J2EE Patterns about Presentation Tier. For this purpose, the Pattern Detection Engine (JPDE) was upgraded with the capability to notice patterns of the presentation tier. For this purpose, the following extensions were applied.

- Addition of Three (3) more Features in the already available Features Catalogue.
- Extension of information in Super parsing Module (JPSP) for the addition in meta-model forth new features.
- Addition of algorithms in Pattern Detection Module (JPDE) for the discovery of newly added features of 5 J2EE Patterns at presentation tier.

The exiting parsing capability of Enterprise Architect (EA) [5, 6, 11] is fine-tuned by using a super parsing module. EA is a well-versed and famous tool for the modeling of software systems [7-9, 14]. This tool is also effectively used to recover design from the source code. However, Enterprise Architect has a weak parsing mechanism and encounters the following deficiencies mentioned below.

Table 8. Deficiencies of Enterprise Architect

1	Resolving Delegation of Cross language artifacts
2	Resolving Association of Cross language artifacts
3	Resolving Association among Function Parameters
4	Resolving Association Return Type Function
5	Resolving Association among Local Variables

6	Resolving Aggregation

More, the EA lacks the following relationships to resolve

- 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

Extended Super Parsing Module (ESPM) and its Approach:

Initially, the raw MDB model was created using EA Tool. The deficiencies of the model were resolved in the form of a Super parsing module. The role of the super parsing module is to enhance the initial mete model created by the EA RDB model. To extend the existing functionality and to cater all the information prevalent to the Presentation tier and detection of the Presentation tier pattern, the Extended Super Parsing Module (ESPM). The ESPM is an extended RDB Model containing the initial RDB model of EA upgraded to a super parsing model and extended capability to have all the information to detect J2EE Presentation Tier Design Patterns.

The existing model JPSP was reinforced to ESPM by introducing definitions of 5 J2EE Patterns relevant to Presentation Tier (cater Presentation layer information). Although some information was already available, however, the process for getting Servlet information features was yet to be proposed. So, features# 44 to 46 (3 features) were introduced in ESPM Module.

The Super Parsing Module is equipped with multiple techniques including regular expressions, parsers multiple languages like HTML, JSPs, XML, and Java, etc. This module performs the following operations as mentioned in Table 10.

Detection of Association	
1	Local variable and resolving their scope
2	Using Symbol table for Type resolution
3	Resolving weak associations
4	Association Through Local Variables
5	Association Through Operation Parameters
6	Association Through Function Return-Type
Detection of Delegation	
1	Detecting Delegation By Call Scope
2	Detecting Delegation Relationship

The ESPM is displayed to J2EE Pattern Detection Tool (JPDT) enhanced with the extra definitions of remaining presentation tier design patterns. This tool has Pattern definitions of Presentation tier and mines through ESPM Module

Extended Visualization Tool Module for Presentation Tier Patterns (EVPM):

The extended visualization module (EVPM) is responsible for the show-case of the Presentation Tier pattern's instances realized from the source code of the enterprise applications. The navigational component supports precise marking of the detected pattern instances within the source code of the applications. Using this capability, the UML of the Pattern Instance It is pertinent to mention that by using the visualization module all the components (that participate in the constitution of Presentation Tier Patterns) can be individually monitored within the

source code. This process enables the dependency analysis and propagation analysis of the source code components.

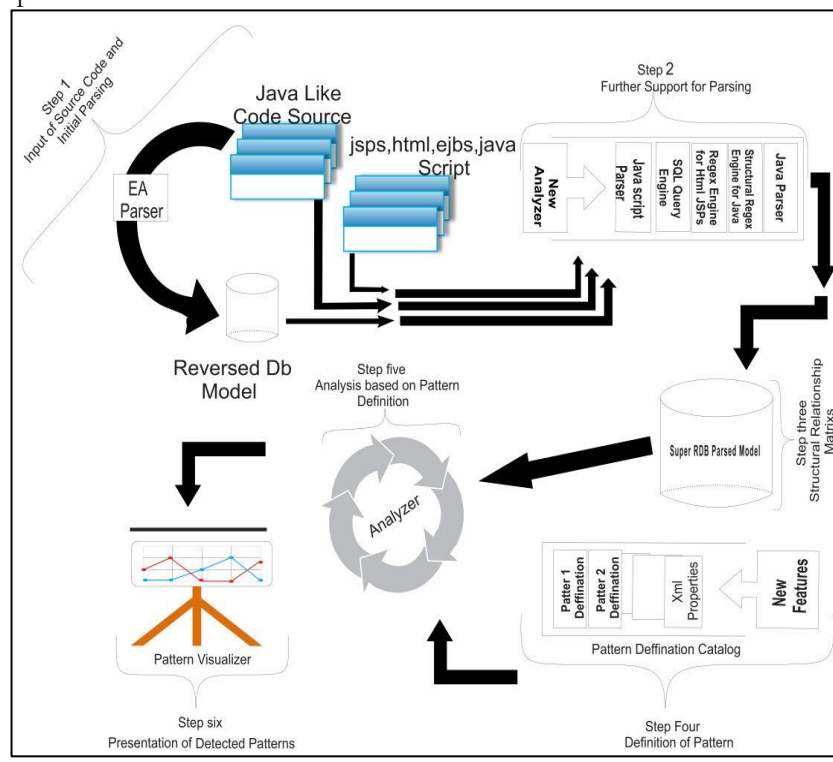

Figure 6. Presentation Tier Design Pattern Detection Approach

Investigation of Approach: Case Study

It is required to validate the proposed methodology for the identification of Presentation Tier Patterns based on extended feature types (Figure7) through reliable and most recent Enterprise Applications. For this purpose, In this section, the evaluation process is performed on reputable medium and large-scale enterprises Applications including Java Pet Store, EJBCA, Apache OFBIZ, Open Brava, and GeoServer, [15-25]. More, the documentation and source code of these ERP applications is available and free to use.

3. Result and discussion

The results of the proposed approach were compared with the existing approach [1] on earlier mentioned open source Enterprise Applications [15-25]. The outcome clearly shows the realization of a complete catalog of Presentation Tier Patterns instances from every application.

It is pertinent to mention that the designated case studies were extensively used in medium and large-scale applications. The manual code inspection of instances of Presentation Tier Patterns is not possible. Keeping this fact in mind, it is ensured that the recovered instances were manually validated.

Extraction of Source Code Metrics and Relationships

Initial stats of the results based on the tool evaluation of the selected case studies on open-sourceEnterprise Applications are shown in Table11. During the process of presentation Tier design pattern recovery, some object-oriented types and interclass relationships are found in the form of classes, Packages, Interfaces, Methods, Attributes, Associations, Generalizations, and Realizations (Shown in Table 12). All these attributes are the building block of the Presentation Tier Design Pattern. Moreover, during this process, multiple cross-language files

and their relationships are also recovered including Java, JSP, HTML, XML, SQL, and property files mentioned in Table 13.

Realization of Presentation Tier Pattern Instances

The tool evaluation results show the realization of Presentation Tier pattern instances from the prescribed open-source Enterprise applications. The outcome of the evaluation is shown in Table 15. The older version of the tool was limited for recovering Presentation tier Patterns, while the present version is capable to recover all the Design Patterns of the presentation tier. More the existence of recovered pattern instances is verified through manual code inspection. We found single instances of Presentation tier Patterns in the source code; this is due to the fewer utilization of specified pattern instances. Moreover, fewer patterns were not realized. However, deep manual examinations we found their definitions but did not qualify for actual pattern definition as prescribed by the sun microsystem.

The recovered pattern instances are thoroughly inspected manually within the source code and found correct. Primarily, found some false positives but all of them were removed when we narrow down the criteria and refine the actual pattern definition with the pattern detection algorithm.

Moreover, we did not find some presentation tier Patterns instances from the selected applications [15-25]. We discovered through manual research that the source code for these patterns didnot match the stated principles offered by the solarmicro system and did not follow the definite structure. Handling Design Patter's Variants is another research dimension. This research focouseson actual definitions of J2EE Design Patterns.

Table 11. Initial Metrics of Selected Software Applications (Case Study)

Source Code Metrics	Open Source Enterprise Applications				
	Open bravo [15]	JPet Store [21]	EJBCA [22]	Geo Server [23]	OFBiz [24]
Application size MB	380	11.1	57.4	104	146
Directories	1,591	378	980	1,040	1,745
Lines of Code (LOC)	434,043	6,573	357,952	192,403	356,474
Blank Lines of Code (BLOC)	44,596	4,603	39,871	28,745	39,221
Physical Executable Lines of Code (PLOC)	306,605	17,891	230,877	98,738	259,761
Logical Executable Lines of Code (LLOC)	221,021	13,957	174,124	74,019	203,697
McCabe VG Complexity (MVG)	38,267	1,796	23,501	13,867	43,723
Code and Comment Lines of Code (CSLOC)	2,109	77	2,241	571	771
Comment Only Line of Code (CLOC)	82,842	14,079	87,204	64,920	57,492
Commentary Words (CWORDS)	508,444	103,222	505,004	276,208	392,418
Header Comment Line of Code (HLOC)	32,930	10,828	20,230	3,778	20,805
Header Commentary Words (HCWORD)	240,627	86,048	122,211	26,577	149,924

Table 12. Metrics of Classes, Objects, and Interclass Relationships

Metrics	Open Source Enterprise Applications				
	Open bravo [15]	JPet Store [21]	EJBCA [22]	Geo Server [23]	OFBiz [24]
Packages	198	128	614	144	276
Total classes	1,987	267	2,121	1,121	1,135
Abstract Classes	83	21	181	64	100
Interfaces	68	63	212	76	90
Methods	14,818	1,955	36,446	9,885	15,544
Attributes	7,232	1,132	13,253	3,275	6,153
Associations	21,662	4,307	158,334	4,826	15,185
Generalizations	1,318	43	1,225	557	707
Realizations	227	29	439	134	263
Total Connections	23,362	4,385	166,742	5,624	22,221

Table 13. Identification of Cross-Language Files

Cross Language Metrics -	Open Source Applications				
	Open bravo [15]	JPet Store [21]	EJBCA [22]	Geo Server [23]	OFBiz [24]
Java Files	2,387	467	3,823	1,413	2,139
XML Files	2,341	97	3252	405	2,732
HTML Files	450	37	554	75	46
JSP Files	1	98	125	146	140
SQL Files	122	5	29	5	11
All Parsed Files	4,746	541	6168	1,669	4,076
Other Files	5,753	206	3,418	3,064	5,813
Total Files	10,499	747	9,586	4,733	9,889
Cross Lang Associations	18,862	3,729	141638	2,199	2,787

Measuring Precision and Recall

To validate the extracted pattern instances, the approach is measured by calculating the results in terms of precision and recall metrics. These metrics help in determining the authenticity of the Design Pattern extraction approach for the Presentation Tier. They were used to examine the quality of the approach by identifying the relevant Presentation Tier Patterns and then calculating the relevant instances that are recovered[25].

However, there are certain shortcomings i.e., in the case of the large source code examination, measuring recall becomes challenging as the manual examination is difficult and time-consuming. Identification of false negatives requirescomparison with valid and reliable benchmarks. Achieving both precision and recall metrics at the maximum level is difficult [26].

The outcome of the case study examination validated the proposed solution from the case study of open-source ERP applications.The detail of Presentation Tier Patterns is given in

Table 15, whereas the detail of recovered Presentation Tier instances along with the false positives and precision in Table 14 and Figure 8 respectively

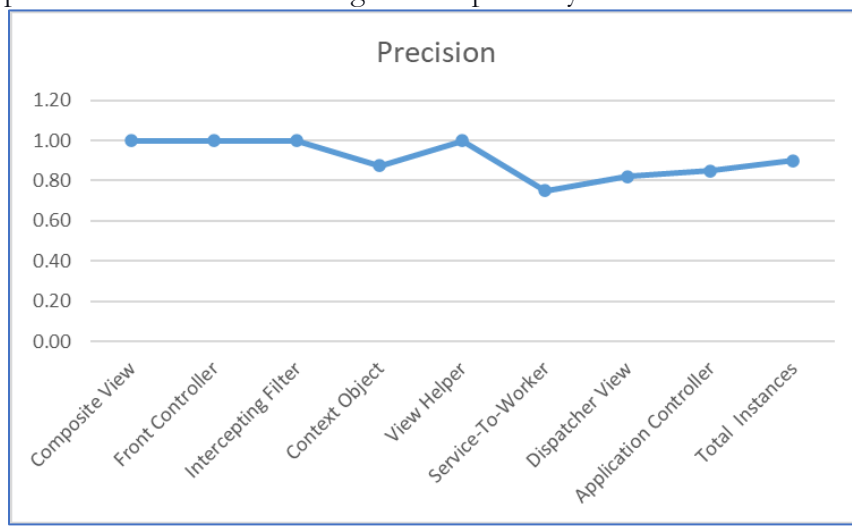

Figure 8. Precision of Presentation Tier Patterns

Table 14. Precision Summary of ERP Application

Enterprise Applications	Instances	False +ves	Precision
Openbravo [15]	39	4	90%
Java Pet Store [21]	38	0	100%
EJBCA [22]	54	3	94%
GeoServer [23]	42	7	83%
OFBiz [24]	43	10	77%
Total Instances	**216**	**24**	90%

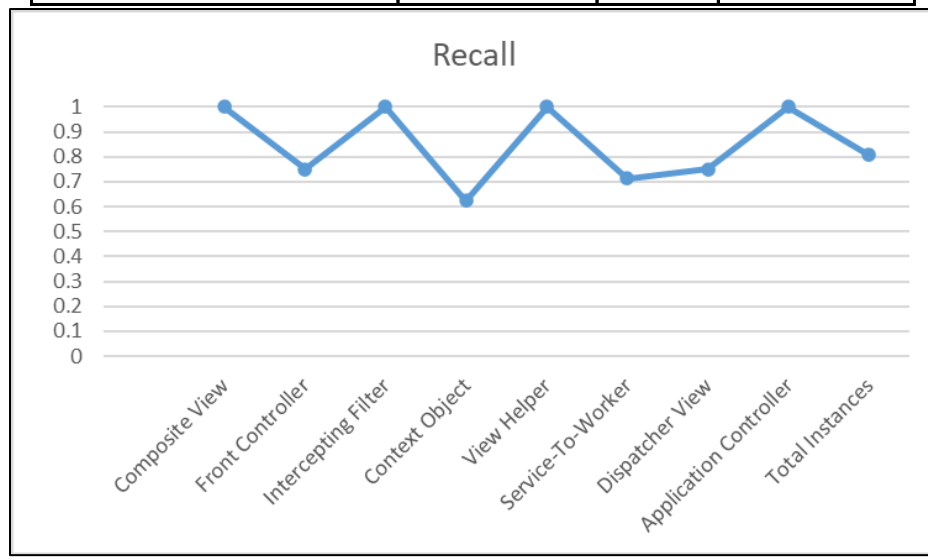

Figure 9. Presentation Tier Pattern's Recall Metrics Java Pet Store [22]

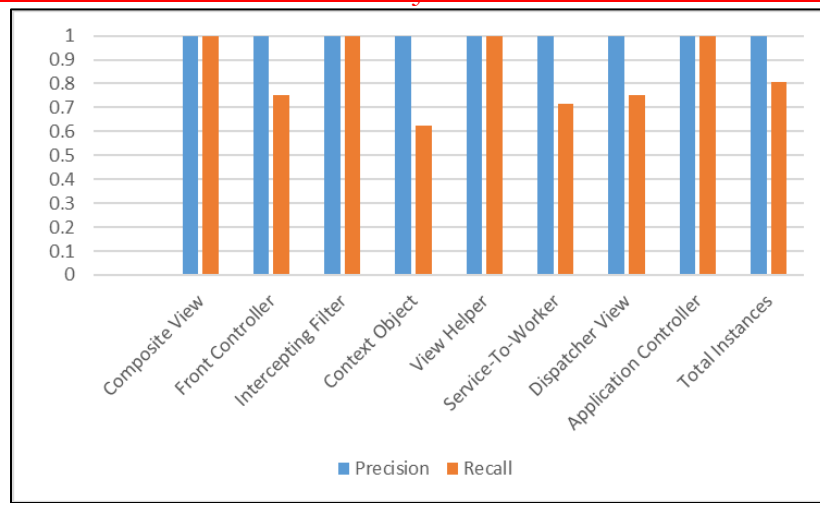

Figure 10. Presentation Tier Pattern's Precision & Recall Metrics Java Pet Store [22]

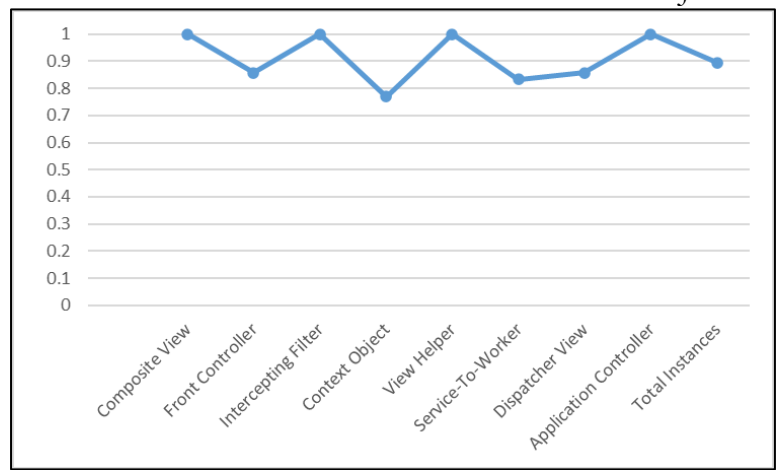

Figure 11. Presentation Tier Pattern's F-Score Metrics Java Pet Store[22]

However, due to the absence of a benchmark, measuring recall is very difficult and manual authentication is cumbersome and extensive especially for large source code applications. We tried to select an application with moderate source code and it contained verified instances of J2EE Design Patterns. J Pet store [21] found a suitable candidate as this is a medium-level application by Sun Microsystems and is enriched with actual instances of J2EE design patterns. The findings of manual inspection of code and recovered pattern instances supported our approach through recall matrices are presented in Figure 9. The comparison of precision and Recall metrics is staged in Figure 10. The F-Score is a measure between precision and Recall. In our case study for JPet Store, the F-score is 0.89, which is quite healthy. The F-Score measure is mounted in Figure 11.

Discussion

The present approach supports the realization of J2EE Design Patterns. These patterns contain cross-language artifacts that require the identification of all cross-language components that participate in the pattern's construct. Initially, the concept of cross-language code analysis and standard for the realization of J2EE Patterns was presented. For this purpose, a catalog for the recovery of J2EE Design Patterns was offered that was capable to realize only 10 J2EE Design Patterns dispersed on all layers of the software application.

However, when we discuss specifically presentation tier design patterns, it is observed that the earlier approach supports the recovery of only three design patterns prevailing to presentation

tier including Composite View, Front Controller, and Intercepting Filter patterns (Sr # 1 to 3 in Table 15). This approach was deficient to provide complete information of the presentation tier revealed by design pattern recovery. As a result, a more comprehensive methodology was needed to ensure the complete recovery of design patterns at the presentation tier.

The existing method is enhanced to recover all Presentation Tier design pattern instances. Initially, the Catalogue of feature types is expanded by three extra features, resulting in an extended catalog of customizable and extendable feature types. Secondly, to extract all the complete artifacts that participated in the definition of presentation level design pattern, the existing module was upgraded in the form of an Extended Super Parsing module (ESPM). Thirdly, the existing pattern detection module JPDT enhanced to extended JPDT i.e., EPDT. In this module, the new pattern definitions were added in the form of a pattern detection algorithm to realize all the instances of presentation design pattern instances within the source code. Last but not least the exiting visualization module is extended to EPVM to show and navigate the recovered presentation tier design pattern instance within the source code.

Table 15. Presentation Tier Design Pattern Instances Extracted (Case Study ERP Applications)

Instances of Presentation Tier J2EE Design Patterns		Open-Source Applications				
		Open-bravo [15]	JPet Store [22]	EJBCA [26]	Geo Server [27]	OFBiz[24]
1	Composite View	11	1	2	13	11
2	Front Controller	2	3	2	1	1
3	Intercepting Filter	2	1	4	1	1
4	Context Object	4	5	11	7	15
5	View Helper	1	12	6	15	1
6	Service-To-Worker	1	5	5	3	1
7	Dispatcher View	7	9	3	1	3
8	Application Controller	11	2	21	1	10
Total Pattern Instances		39	38	54	42	43

Threats to Validity

This section addresses issues about the proposed approach's acceptability in terms of its validity, which refers to validity which means the confirmation of the approach through empirical results and demonstrating that the suggested research is a substantial contribution with proof of concept.

Internal validity metrics ensure the technique validated by tools or methodology is reliable [28].The current approach aids in the detection of multilingual J2EE Design Patterns' presentation tier. The prior method could only discover a few patterns related to presentation tier patterns; nevertheless, all of the current pattern definitions are an extension of past research.Standard pattern definitions and their related attributes are derived from authentic and dependable resources utilizing an adaptive and expandable feature to avoid risks to internal validity[2, 25, 29-31].

This approach is implemented in the J2EE Pattern Detection Tool, which is capable of extracting Patterndescriptionsfrom the source code of designated applications. The results validate the approach through open-source ERPs [15-25] . However, manual inspection of the outcome is needed to avoid false positives. In this regard, community participation is necessary

to strengthen the results and reduction of the effect of biases. For further evaluation, the results shall be available on the GitHub repository. As already discussed, the previous approaches can't support the detection of presentation tier design patterns in a multilingual environment. The external validity demands generalization of approach on large scale. For this purpose, we initial tested our system on JPET Store [22] by sun microsystems, then we further evaluated our approach on famous and commonly used medium/ large scaled ERPs [15-25] .All of these applications are open source and their documentation is available for further validation [32-36]. All the extracted pattern instances for the presentation tier are manually inspected and found correct, however, generalization in terms of precision and recall for all presentation tier pattern instances is quite challenging. The pattern definitions are customizable and extendable to accommodate for any variation in existing patterns or any addition of new pattern definitions. This nature of feature types generalizes the approach to accommodate any kind of pattern definitions and is scalable to detect them within the source code of multiple object-oriented languages.

4. Conclusion

In this research, customizable and extendable definitions are proposed that enable the extraction of presentation tier information in the form of design pattern recovery. The approach is validated from a reliable open-source multilingual ERP application. The approach is customizable and extendable to accommodate variants and new design definitions. The technique is validated on J2EE Design Patterns detection. At present we are working on the detection of patterns of integration tier and business application tier. Moreover, we are acting on the detection of recurring design definitions and variants handling.

Acknowledgement.
We would like to thank with deep sense of gratitude to **Dr. ZaighamMushtaq** for his keen interest, inspiring guidance and endless support with our work at all stages.

Author's Contribution.

Designing The Experiment: ZaighamMushtaq, Ghulam Rasool
Performed The Experiments: ZaighamMushtaq, Ghulam Rasool
Analyzing The Data: ZaighamMushtaq
Code, Designed the Software or Performed the Computation Work
ZaighamMushtaq, Ghulam Rasool
Work or Revised It Critically for Important Content:
ZaighamMushtaq

Conflict of interest. Authors has no conflict of interest for publishing this manuscript in IJIST.
Project details. The aim of this research is the reusability of exiting code and design pattern recovery of lagecy application. The ultimate goal of this project is to create automated documentation of existing legacy code/

References

1 Mushtaq, Z., Rasool, G., and Shahzad, B.: '*Detection of J2EE Patterns based on Customizable Features*', INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 2017, 8, (1), pp. 361-376

2 Alur, D., Malks, D., Crupi, J., Booch, G., and Fowler, M.: '*Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies. Mountain View*, CA, USA: Sun Microsystems', in Editor (Ed.)^(Eds.): 'Book Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies. Mountain View, CA, USA: Sun Microsystems' (Inc, 2003, edn.), pp.

3 Aniche, M., Yoder, J., and Kon, F.: '*Current challenges in practical object-oriented software design*', in Editor (Ed.)^(Eds.): 'Book Current challenges in practical object-oriented software design' (IEEE, 2019, edn.), pp. 113-116

4 Fowler, M.: '*Patterns of enterprise application architecture.-Addison-Wesley Longman Publishing Co*', 2002

5 Tiwari, K.: 'Study and Assessment of Reverse Engineering Tool', 2020

6 Belfadel, A., Amdouni, E., Laval, J., Cherifi, C.B., and Moalla, N.: '*Towards software reuse through an enterprise architecture-based software capability profile*', Enterprise Information Systems, 2020, pp. 1-42

7 Afzal, K.: 'Formal Verification of Software Models in MDE', 2017

8 Ibrahim, L.M., and Ibrahim, K.A.: '*Constructing an Add-in Tool for Enterprise Architect v7. 5 To Measure the Quality of Object Oriented Design (Class Diagram)*', International Journal of Computer Science and Information Security, 2015, 13, (7), pp. 72

9 Gahalaut, A.K., and Khandnor, P.: '*Reverse engineering: an essence for software re-engineering and program analysis*', International Journal of Engineering Science and Technology, 2010, 2, (06), pp. 2296-2303

10 Mushtaq, Z.: '*Multilingual Source Code Analysis for Recovery of J2EE Environment*', 2017

11 Fekete, A., and Cserép, M.: '*Incremental Parsing of Large Legacy C/C++ Software*', in Editor (Ed.)^(Eds.): 'Book Incremental Parsing of Large Legacy C/C++ Software' (2018, edn.), pp. 51-54

12 Fowler, M.: '*Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch'* (Addison-Wesley, 2012. 2012)

13 Rubis, R.: '*Patterns for Enterprise Application Design and Development*', Florida Atlantic University, 2017

14 Mark, C.: 'Sun Certified Enterprise Architect For Java Ee Study Guide, 2/E' (Pearson Education India, 2010. 2010)

15 Ortiz, J.C.V.: '*DiseÃ±o de un software que integre una tienda online con Openbravo ERP*', Revista Matices Tecnológicos, 2018, 7

16 Jain, A., Gupta, S., Vyas, M., Pathy, D., Khare, G., Rajan, A., and Rawat, A.: '*Open source EJBCA public key infrastructure for e-governance enabled software systems in RRCAT*': 'ICT Based Innovations' (Springer, 2018), pp. 127-139

17 Ryoo, H.-G., Kim, S., Kim, J.-S., and Li, K.-J.: '*Development of an extension of GeoServer for handling 3D spatial data*', in Editor (Ed.)^(Eds.): 'Book Development of an extension of GeoServer for handling 3D spatial data' (2017, edn.), pp. 6

18 AS, M.P.: '*ERP OPEN SOURCE APACHE OFBIZ', Jurnal E-Komtek (Elektro-Komputer-Teknik)*, 2018, 2, (2), pp. 129-133

19 Aversano, L., Guardabascio, D., and Tortorella, M.: '*Analysis of the documentation of ERP software projects', Procedia computer science*, 2017, 121, pp. 423-430

20 Rychkova, I., Regev, G., Le, L.-S., and Wegmann, A.: '*From business to IT with SEAM: The J2EE Pet Store example*', in Editor (Ed.)^(Eds.): 'Book From business to IT with SEAM: The J2EE Pet Store example' (IEEE, 2007, edn.), pp. 495-495

21 Schuts, M.: '*Industrial experiences in applying domain specific languages for system evolution*', [Sl: sn], 2017.

22 Technology, O.: '*Java Pet Store*', 2021, 1.3.1_02 https://www.oracle.com/java/technologies/petstore-v1312.html

23 Kalyanam, R., Zhao, L., Song, C., Biehl, L., Kearney, D., Kim, I.L., Shin, J., Villoria, N., and Merwade, V.: '*MyGeoHub—A sustainable and evolving geospatial science gateway', Future Generation Computer Systems*, 2019, 94, pp. 820-832

24 OFBiz, A.: 'Apache OFBiz', 2021 https://blogs.apache.org/ofbiz/entry/apache-ofbiz-news-may-2021

25 Crupi, J., and Baerveldt, F.: '*Implementing Sun Microsystems' Core J2EE Patterns*', Compuware White Paper, 2004

26 AB, P.S.: '*EJBCA Enterprise*', 2021 https://www.primekey.com/products/ejbca-enterprise/
27 Foundation, O.S.G.: 'GeoServer', 2021 http://geoserver.org/
28 Elish, M.O., and Mohammed, M.A.: '*Quantitative analysis of fault density in design patterns: An empirical study', Information and Software Technology,* 2015, 66, pp. 58-72
29 Crawford, W., and Kaplan, J.: '*J2EE Design Patterns: Patterns in the Real World*' (" *O'Reilly Media, Inc.*", 2003. 2003)
30 Alur, D., Crupi, J., and Malks, D.: '*Core J2EE patterns: best practices and design strategies'* (Gulf Professional Publishing, 2003. 2003)
31 Johnson, R., and Hoeller, J.: '*Expert one on one J2EE development without EJB*' (John Wiley & Sons, 2004. 2004)
32. http://geoserver.org/download/, accessed 01102016 2016
33. http://www.openbravo.com/product-download/, accessed 01102016 2016
34. https://www.primekey.se/technologies/products-overview/ejbca-enterprise/
35. https://www.ejbca.org/index.html, accessed 01102016 2016
36. http://ofbiz.apache.org/download.html; , accessed 01102016 2016