Ontology Driven Software Development for Automated Documentation

Webology, Volume 15, Number 2, December, 2018

Ontology Driven Software Development for Automated Documentation

M.P.S. Bhatia
Division of Computer Engineering, Netaji Subhas Institute of Technology, Delhi, India. ORCID.
E-mail: mps.bhatia@nsit.ac.in

Akshi Kumar
Department of Computer Science and Engineering, Delhi Technological University, Delhi, India.
ORCID. E-mail: akshikumar@dce.ac.in, akshi.kumar@gmail.com

Rohit Beniwal
Department of Computer Science and Engineering, Delhi Technological University, Delhi, India.
ORCID. E-mail: rohitbeniwal@yahoo.co.in

Received September 10, 2018; Accepted December 25, 2018

Abstract

Recent outsourcing /off-shoring software development practices testify that any development done without a proper sharing mechanism leads to the generation of inconsistent information, which further results in an undesired, error-prone software. Further, with the business process automation, a significant way to minimize human effort involves various development, support and maintenance activities to reuse available information. Thus, reusing and sharing information in a standardized way is the key operative challenges which foster the need to identify and exploit novel knowledge-based frameworks. The proposed research provides a tool-based solution to automate the software documentation process using ontologies. This multi-phase framework has overall six phases where each phase output contributes to the final automated documentation. To evaluate the extent of automated documentation it is compared using free and open source software known as WCopyfind to the existing manual documentation for a Result Management System case study. Preliminary results show a highest automation of 60 percent, which is clearly noteworthy.

Keywords

Automatic documentation; Ontology; Ontology driven; Semantic Web; Software's documentation; Software engineering

Introduction

The “Web/ Mobile Apps,” “Agile Development Practices,” “Big-Data,” “Security,” “Cloud Computing,” “IoT,” “Open Source,” “Customer-first Design” are some of the key terms that characterize the latest trends in the software development technology. It has been established across pertinent literature that a well-balancing act between the Iron Triangle or the Project Triangle representing the triple constraints of Time-Cost-Quality along with an added scope and sustainability dimensions can lead to successful software development and delivery (Kumar & Gupta, 2018). The upsurge in economic globalization has compelled organizations to gain a competitive advantage by cutting their costs, optimizing efficiency, and at the same time provide superlative customer service. Outsourcing /off-shore software development practices have proven to be vital, valuable and profitable for many organizations worldwide as geographically distributed teams can offer huge benefits in terms of efficiency and cost savings. Moreover, the ability to choose team members with the best skills by passing the location-based hiring pool can lead to more focused and strategic development. In this ‘Think Global, Code Local’ approach, time-zone coordination, diverse development culture collaboration, team unity around a team charter, strong communication and knowledge sharing technologies arise as significant challenges. Pertinent studies have indicated that any development work that is done with a lack of sharing mechanism leads to the generation of inconsistent information, which further results in an undesired software (Bhatia et al., 2014; 2016b). Moreover, with the business process automation, a significant way to minimize human effort involves various development, support and maintenance activities to reuse available information. Thus, reusing and sharing information in a standardized way is the key operative challenges which foster the need to identify and exploit novel frameworks (Anunobi et al., 2008; Knight & King, 2010).

The above challenge leads to the integration between research fields of Semantic Web (SW) technologies and Software Engineering (SE) (Zhao et al., 2009; Bhatia et al., 2016a; Gašević et al., 2009; Bhatia et al., 2016c) because “Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries” (“W3C”, n.d.a; “W3Cb”, n.d.b). Alternatively, Software Engineering is “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software” (IEEE Standards Coordinating Committee, 1990). This integration opened new avenues for researchers to look at various issues and challenges that got generated due to to the amalgamation of SW and SE (Jenkins, 2008). Among such issues, one is ontology driven software development for automated documentation (Bhatia et al., 2016a; Bhatia et al., 2015).

Documentation is an essential part of the software as it is helpful in proper communication and sharing of information. It consists of various manuals such as Software Requirements Specification (SRS) generated during requirement engineering phase, Software Design

Document (SDD) produced during the design phase, source code listing created during the implementation phase, and test data and test result manufactured during the testing phase. Good quality of documentation provides a great help in maintenance as well as in reverse engineering phase. However, in many cases, it is seen that documentation is not built with right enthusiasm. The documentation is either completely or partially ignored, which is mainly because of cost and schedule constraint along with the complexity of software system involved (Tang et al., 2006; Kruchten et al., 2009). The quantity and effort estimates for the right amount of documentation have been primary concerns too. In IT industry where Agile development methodology is quite popular, its focus is more on people and interactions between them; software deliverable rather than lengthy documents, customer collaboration throughout the development cycle, and on quick responses to change (Cockburn, 2002; Martin, 2002). In such a collaborative development environment where multiple cross-functional teams are working, automation of software's documentation is a primary need to meet the requirements of adopted agile paradigm (Foerstel, 2002; American Library Association, 2013).

Hence, the work presented in this paper provides a solution where an ontology driven software development framework for automated documentation is modeled, implemented, and analyzed with a case study. The solution caters few of the requirements of agile paradigm-setting and aids in its convenient practical adoption.

In ontology driven software development for automated documentation, a different approach is adopted for software development as compared to our existing traditional approaches. Here we develop the software using ontologies. "Ontology is formal and explicit specification of a shared conceptualization" (Studer, 1998). Ontologies are built to model a domain and support reasoning over the concepts. Ontology Engineering in Semantic Web is primarily supported by languages such as RDF, RDFS and OWL (Ding, 2007).

To accomplish ontology driven software development for automated documentation, first of all, a multi-phase framework is proposed which is then exhibited using a case study. This multi-phase framework has overall six consecutive phases, namely domain ontology phase, software requirements specification ontology phase, design phase, source code phase, source code listing ontology phase, and test cases ontology phase where the output of one phase becomes input to next phase. Each of the phases has its output which contributes towards final automated documentation. Further, to evaluate and verify the proposed framework for automation, we compare the automated documentation to the existing manual documentation of case study using an open source software. This open source software finds out the similarities between two documents which in turns describes that what percentage of automation is achieved. As a result of our work, we achieved maximum 60 percent automation when this open source software is tuned to one of the appropriate settings. Another advantage of this approach is that it generates the documentation in human as well as in machine-understandable form (American Library

Association Council, 1939). As it is also available in the machine-understandable form, documentation is free from inconsistencies and ambiguities (Liu, 2005; Knox, 2015).

The rest of the paper is organized as follows: section 2 discusses the related work; section 3 introduces the framework followed by section 4 which expounds the implementation of the framework, and section 5 presents the result and analysis. Finally, section 6 concludes the paper and provides direction for future work.

Background

To improve the overall description of documentation, de Graaf (2011) and Tang et al. (2011) made an annotating semantic wiki page with the help of lightweight Software Engineering ontology. This lightweight Software Engineering ontology had limited no. of classes and properties. Further, properties comprise of Dublin core data properties to allow specification of metadata. Both the authors created the wiki page with the help of ArchiMind and OntoWiki Semantic Wiki in which the annotated text can be searched. The only difference between their work is that Graaf provided the documentation software requirements and architecture design part only. So far, none of them generated any documentation file. The annotated text in ArchiMind and OntoWiki Semantic wiki is like a tooltip for particular selected text.

López et al. (2012) developed ontology based tool known as Toeska Rationale Extraction (TREx) tool for automating the software architecture documentation part only. They implemented the TREx on a case study and compared its outcome vs. plain text documents, and it was found that humans are poorer than TREx in identifying rationale, but are better at dealing with "generic" components.

de Graaf et al. (2014) also provided an exploratory study on ontology engineering approach for software architecture documentation. They described an approach that uses typical questions for elicitating and constructing an ontology for software architecture documentation. Along with that, they also depicted eight contextual factors that influence the acquisition of typical questions, which are asked from architectural knowledge users. Moreover, they applied this approach on a case study showing how it can be used for obtaining and modeling architectural knowledge needs.

Koukias et al. (2015) developed an ontology-based model to represent the technical documentation content towards using it to optimize the performance of the asset. Furthermore, Koukias & Kiritsis (2015) discussed a step-by-step ontology-based approach for modeling technical documentation to optimize asset management using rule-based mechanism.

Bhatia et al. (2015) discussed ontology-based framework for automatic software's documentation where they have just provided the framework, however, to the best of our knowledge, the framework is not realized or implemented so far. This ontology driven software development for automated documentation is also depicted as an open issue problem by Bhatia et al. (2016a) in their literature survey article.

Therefore, as far as ontology driven automated documentation is concerned, Authors' de Graaf, (2011) and Tang et al. (2011) work does not yield any documentation file. While authors' López et al. (2012) and de Graaf et al. (2104) work is limited to architecture documentation only and that too without specifying what percentage of automation is achieved. Moreover, Authors' Koukias et al. (2015) and Koukias and Kiritsis (2015) work is also restricted for modeling purpose only. Author's Bhatia et al. (2015) work is restrained to providing framework only, missing the actual realization or implementation. However, in this paper, we are extending their work by providing ontology driven software development approach for automated documentation that contains all the requisite details of a software system and not just architecture information. Further, we are also generating the documentation file in human as well as in machine-understandable form followed by its comparison with a case study to show what percentage of automation is achieved.

Framework

The proposed multi-phase Framework for ontology driven software development for automating the documentation process is divided into six phases, namely domain ontology phase, software requirements specification ontology phase, design phase, source code phase, source code listing ontology phase, and test cases ontology phase. Figure 1 depicts this framework.

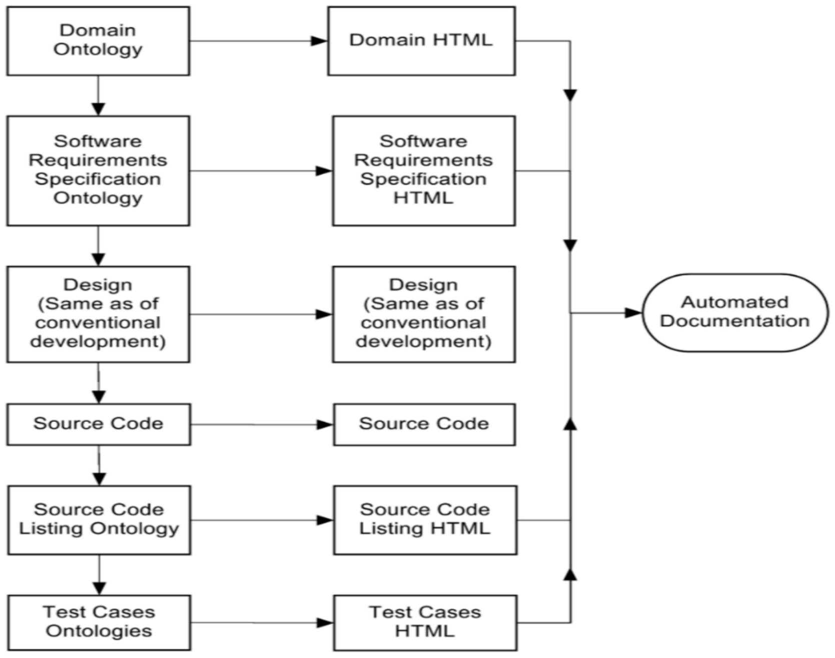

Domain Ontology
Domain HTML
Software Requirements Specification Ontology
Software Requirements Specification HTML
Design (Same as of conventional development)
Design (Same as of conventional development)
Automated Documentation
Source Code
Source Code
Source Code Listing Ontology
Source Code Listing HTML
Test Cases Ontologies
Test Cases HTML

Figure 1. Framework for ontology driven software development for automated software's documentation

The following sub-sections discuss each phase in detail.

1. Domain Ontology Phase

In this first phase, we develop a domain ontology, which is used to capture key concepts of the domain under consideration. This domain ontology encapsulates overall domain based knowledge for a real-world scenario or problem. The domain ontology is typically built from scratch, though few Machine Learning (ML) techniques can be used to automate it. However, 100 percent automation is not possible because of several reasons. One reason could be the complex nature of the real-world problem. For e.g. in case of a complex real-world problem, we may explicitly require domain experts who can help us in understanding various domain related terminologies. Without domain experts, understanding different terminologies such as synonym, homonym, context-dependent interpretation, etc. may lead to ambiguity. Another reason is that ontology development is an iterative process which goes through many cycles of revisions and refinement before it is finally shaped and can be used. Moreover, after each cycle, some human intervention is required for technical evaluation of developed ontology to assess its quality. At the end of all the cycles, the built domain ontology is available in a machine-understandable form which is then converted into domain HTML. This is primarily done to store the domain knowledge into a human readable/ understandable form for future use. Thus, this phase creates and supports understandability in both forms viz. machine-understandable as well as in human-understandable form.

2. Software Requirements Specification Phase

The second phase requires an extension of domain ontology to create Software Requirements Specification (SRS) ontology. The domain ontology serves as the genesis for SRS ontology. IEEE has defined guidelines and standard to organize an SRS document (IEEE Computer Society, 1998a; IEEE Computer Society, 1998b). Based on it, the domain ontology is reconstructed for SRS ontology keeping in mind that it adheres to all the guidelines specified by IEEE. Once this SRS ontology is developed, it is also converted to SRS HTML which can be shared among different geographically and virtually located development teams.

3. Design Phase

In the third phase, we develop the design document known as Software Design Document (SDD) which is same as used in conventional development methodology.

4. Source Code Phase

This phase can also be termed as implementation phase and requires the development and implementation of source code or the actual program fulfilling all requirements laid down by

SRS ontology. That is, in this phase, we perform all frontend and backend related activities to fulfill each and every enlisted requirement.

5. Source Code Listing Ontology Phase

In this fifth phase, a source code listing ontology is built, which enables us to document all the of classes/ variables/ functions used in the source code. Generally, the names of all the components along with their purpose in source code listing document are written. This documentation allows a better understanding of the source code for maintenance purpose. Once this ontology is constructed, we convert it into its HTML form.

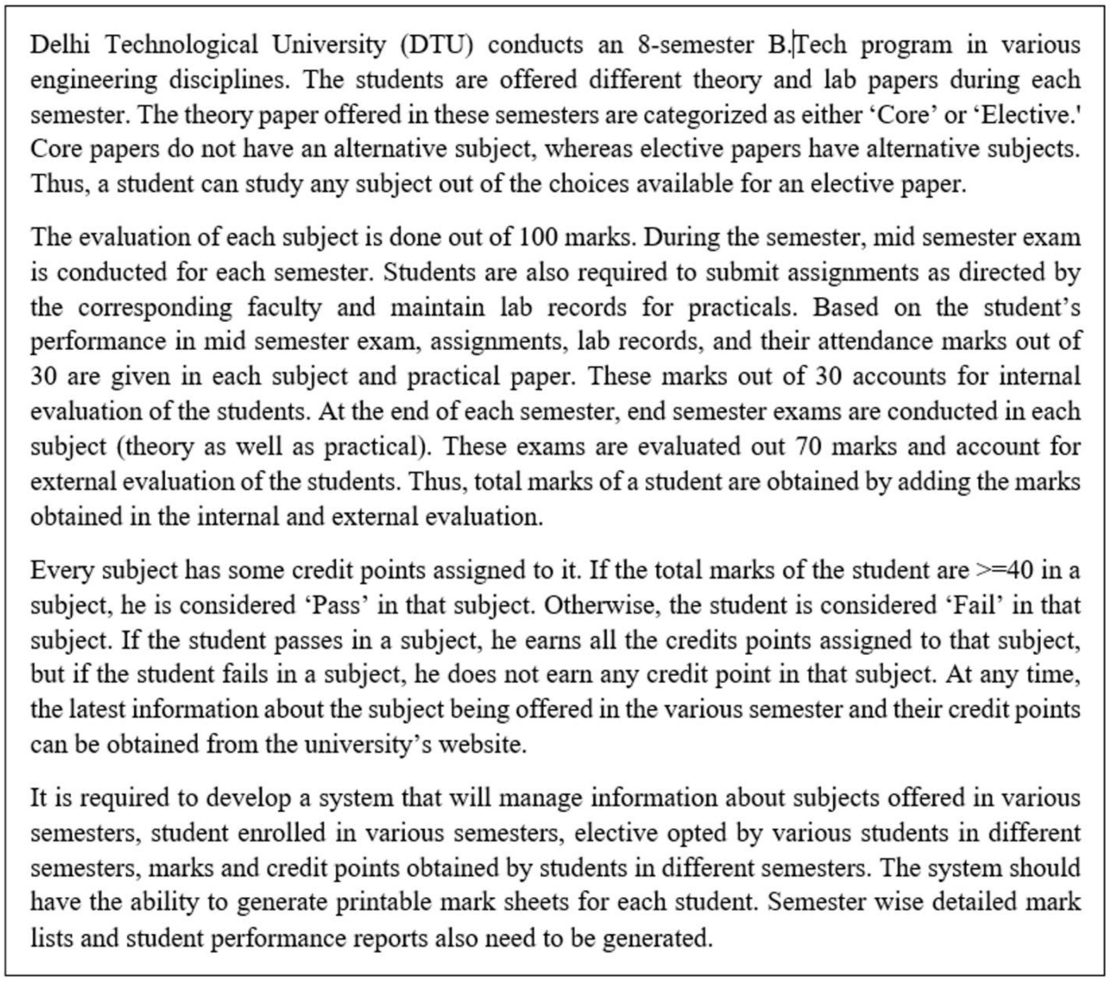
6. Test Cases Ontology Phase

In this phase, the test cases ontologies are developed and converted into HTML form. This phase ensures that the developed software is verified and validated as per SRS and customer expectations.

Thus, as an output from these phases, we have domain HTML, SRS HTML, source code listing HTML, and test cases HTML which consolidate to automate the software documentation process. The next section illustrates the implementation of this proposed framework.

Implementation

To clearly illustrate the effectiveness of the proposed framework, a case study on Result Management System (RMS) which is a software developed for Delhi Technological University (DTU) is presented. It is a management information system that is developed to streamline and accelerate the result preparation and management process. It is used to manage DTU’s day- to-day result based activities such as student details registration, student course registration, student subject details, student marks in each subject, generation of semester-wise and consolidated mark sheets, amongst others. It also generates summary reports regarding student information, semester-wise mark lists, and performance reports. The snapshot of problem statement as taken from case study is shown below in Figure 2.



Delhi Technological University (DTU) conducts an 8-semester B.Tech program in various engineering disciplines. The students are offered different theory and lab papers during each semester. The theory paper offered in these semesters are categorized as either 'Core' or 'Elective.' Core papers do not have an alternative subject, whereas elective papers have alternative subjects. Thus, a student can study any subject out of the choices available for an elective paper.

The evaluation of each subject is done out of 100 marks. During the semester, mid semester exam is conducted for each semester. Students are also required to submit assignments as directed by the corresponding faculty and maintain lab records for practicals. Based on the student's performance in mid semester exam, assignments, lab records, and their attendance marks out of 30 are given in each subject and practical paper. These marks out of 30 accounts for internal evaluation of the students. At the end of each semester, end semester exams are conducted in each subject (theory as well as practical). These exams are evaluated out 70 marks and account for external evaluation of the students. Thus, total marks of a student are obtained by adding the marks obtained in the internal and external evaluation.

Every subject has some credit points assigned to it. If the total marks of the student are >=40 in a subject, he is considered 'Pass' in that subject. Otherwise, the student is considered 'Fail' in that subject. If the student passes in a subject, he earns all the credits points assigned to that subject, but if the student fails in a subject, he does not earn any credit point in that subject. At any time, the latest information about the subject being offered in the various semester and their credit points can be obtained from the university's website.

It is required to develop a system that will manage information about subjects offered in various semesters, student enrolled in various semesters, elective opted by various students in different semesters, marks and credit points obtained by students in different semesters. The system should have the ability to generate printable mark sheets for each student. Semester wise detailed mark lists and student performance reports also need to be generated.

Figure 2. Problem statement for Result Management System as taken from case study

The initial software was developed using conventional methodologies such as Unified Modeling Language (UML) and Object-Oriented programming (OOP) language where the documentation is manually done. We show that using the proposed ontology driven approach automation of documentation is achievable with motivating and beneficial preliminary results. The framework described in the previous section is realized using Protégé version 5.1.0 tool and its plug-ins such as OntoGraf 2.0.3, BrowserView (OWLDoc) 3.0.3, Pellet Reasoner 2.2.0, and HermiT Reasoner 1.3.8. "Protégé is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies" (Stanford Center for Biomedical Informatics Research, 2016). Pellet and HermiT reasoners are used to ensure that developed ontologies are free from any kind of inconsistencies (Stanford Center for Biomedical Informatics Research, 2008; University of Oxford, n.d.; Motik et al. 2008). To be specific, domain ontology, software requirements specification ontology, source code listing ontology, and test cases ontologies are developed using Protégé. Plugin OntoGraf is used to generate graphs of the domain ontology, software requirements specification ontology, source code listing ontology, and test cases ontologies (Falconer, 2013). Plugin OWLDoc is utilized to produce HTML code for all these ontologies (Drummond et al., 2012; Drummond, n.d.) and their reasoning is checked using Pellet and HermiT reasoners. The following subsections elaborate phase wise implementation details of the proposed framework:

1. For Domain Ontology Phase

The domain ontology for RMS is built from scratch by understanding it as a real-world problem with a definitive goal, requirements, and activities. The ontology has 23 classes, 4 object properties, 22 data properties, 2 annotation properties, 23 individuals, and 4 data types with a total of 78 entities. Table 1 lists all classes, object properties, data properties, annotation properties, individuals, and data types belonging to domain ontology.

Table 1. List of classes, object properties, data properties, annotation properties, individuals, and data types belonging to domain ontology

Classes	1. owl:Thing 2. "Domain_(Requirements)" 3. Availability 4. Communication_interfaces 5. Design_Constraints 6. External_Interface 7. External_Interface_Requirements 8. Functional_Requirements	9. Functions 10. Hardware_Interfaces 11. Logical_Database_Requirements 12. Maintainability 13. Non_Functional_Requirements 14. Other_Requirements 15. owl:Nothing 16. Performance_Requirements	17. Portability 18. Reliability 19. Result_Management_System 20. Security 21. Software_Interfaces 22. Software_System_Attributes 23. User_Interfaces
Object Properties	1. accessFollowingScreen 2. defineRoles	3. hasAccess	4. inturnAccess
Data properties	1. Address 2. Batch_year 3. Branch 4. Category/_Type 5. Credit 6. Email_id 7. External_marks 8. Internal_marks	9. Name 10. Password 11. Phone_no. 12. Role 13. Semester_number 14. Student_enrolment_number 15. Student_name 16. Student_registration_number	17. Student_roll_no. 18. Subject_code 19. Subject_name 20. Total_marks 21. Username 22. Year_of_admission
Annotation Properties	1. rdfs:comment	2. rdfs:label	
Individuals	1. Admin 2. DEO 3. Faculty 4. Generate_marksheet 5. Login_screen 6. Maintain_marks_details 7. Maintain_student's_subject_details 8. Maintain_student_details 9. Maintain_subject_details 10. Marks_entry_parameters_screen 11. Marks_entry_screen	12. Marks_list_report_parameters_screen 13. Marksheet_parameters_screen 14. MEO 15. Print_marksheet 16. Student's_subject_choice_information_screen 17. Student's_subject_choice_list_report_parameters_screen 18. Student's_subject_choice_parameters_screen	19. Student_information_parameters_screen 20. Student_information_screen 21. Student_list_report_parameters_screen 22. Subject_information_parameters_screen 23. Subject_information_screen
Data Types	1. Integer 2. rdf:PlainLiteral	3. xs:short	4. xsd:string

Figure 3 shows the graph of domain ontology generated using OntoGraf. To easily understand the nodes, classes and instances, and their relationship with each other, we explicitly add a representation of the node and arc types used within the graph. The classes and individuals are all represented using a rectangular shape. However, a yellow colored circle on the top left side of rectangle depicts that it is a class and a purple colored diamond on the top left side of rectangle exhibits that it is an instance. A plus sign within the node indicates that it can be further expanded. For easy understanding, a sample graph is shown in the following Figure 3.

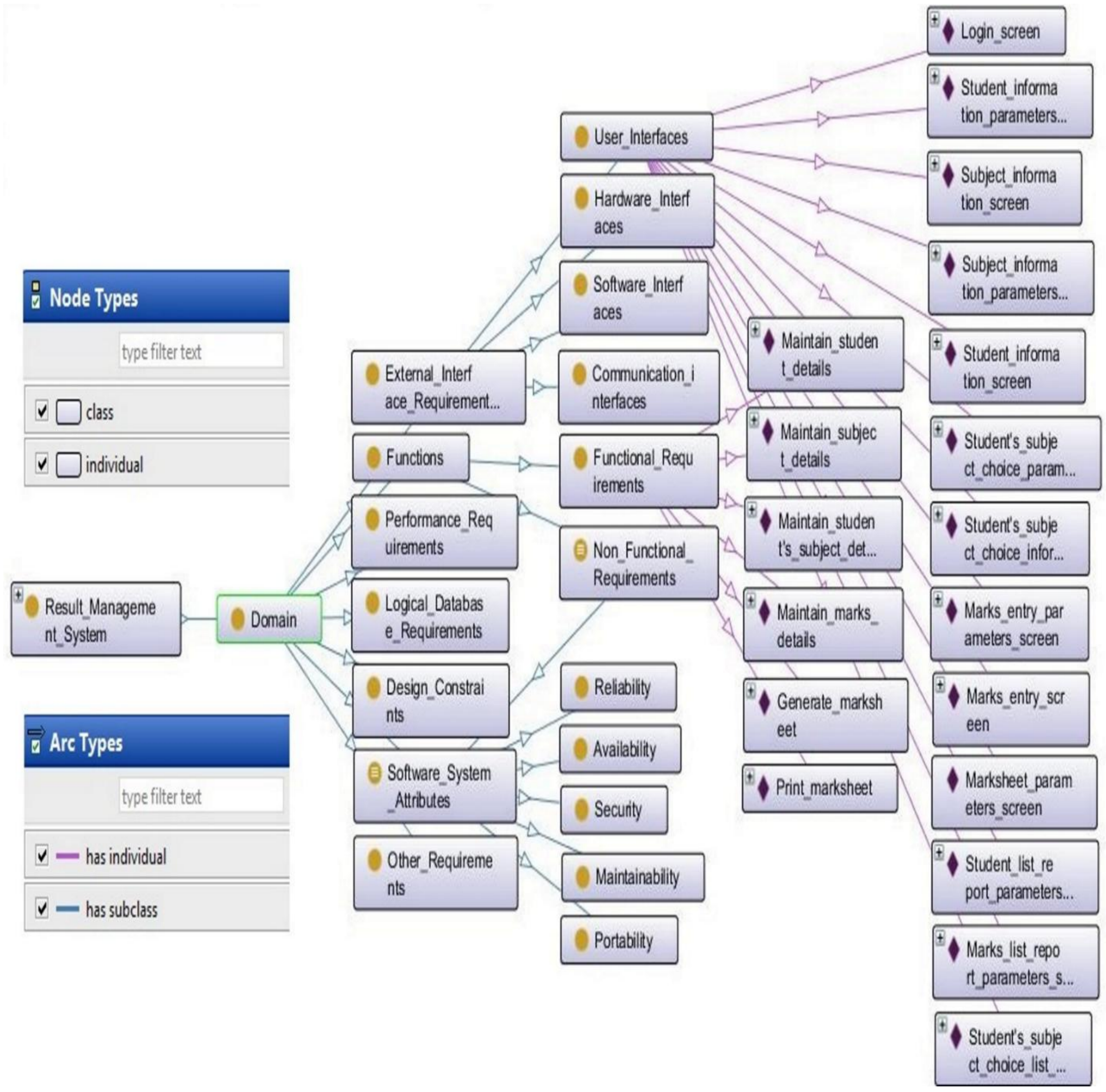
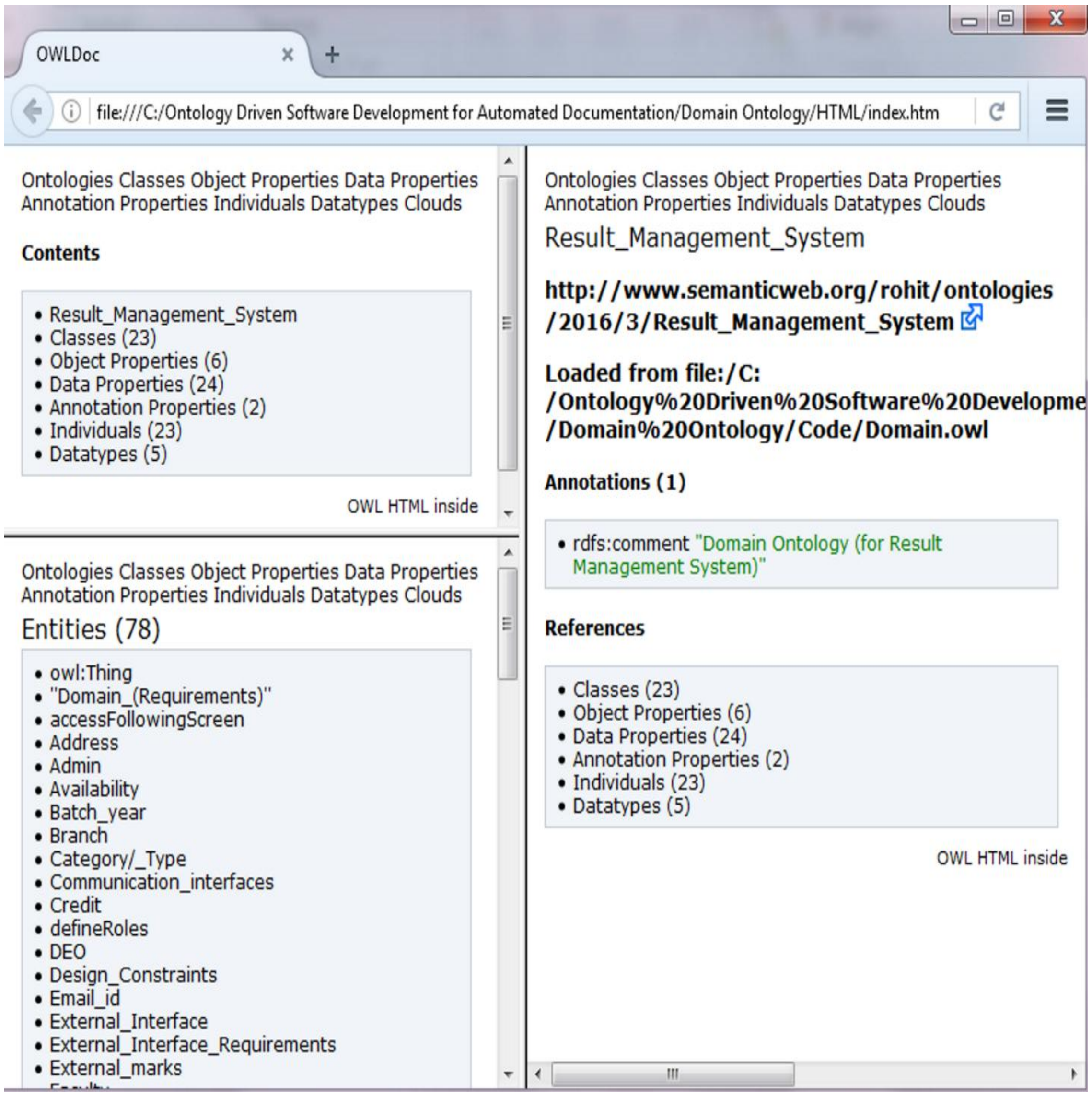

Node Types
type filter text
class
individual
Arc Types
type filter text
has individual
has subclass
Result_Management_System
Domain
External_Interface_Requirement...
Functions
Performance_Requirements
Logical_Database_Requirements
Design_Constraints
Software_System_Attributes
Other_Requirements
User_Interfaces
Hardware_Interfaces
Software_Interfaces
Communication_interfaces
Functional_Requirements
Non_Functional_Requirements
Reliability
Availability
Security
Maintainability
Portability
Maintain_student_details
Maintain_subject_details
Maintain_student's_subject_det...
Maintain_marks_details
Generate_marksheet
Print_marksheet
Login_screen
Student_information_parameters...
Subject_information_screen
Subject_information_parameters...
Student_information_screen
Student's_subject_choice_param...
Student's_subject_choice_infor...
Marks_entry_parameters_screen
Marks_entry_screen
Marksheet_parameters_screen
Student_list_report_parameters...
Marks_list_report_parameters_s...
Student's_subject_choice_list_...

Figure 3. Graph of domain ontology with the explicitly added representation of the node and arc types

Figure 4 shows the HTML of the index page of the domain ontology. This HTML generated using the OWLDoc plugin is in a human-understandable form which makes it viable for sharing amongst geographically located development teams. Each class, object property, data property, annotation property, individual, data type, and the entity in the HTML has a hyperlink which contains its further explanation. For example, figure 5 shows the HTML of domain ontology when its classes hyperlink is clicked, which then gets expanded in the right-side column.


OWLDoc
file:///C:/Ontology Driven Software Development for Automated Documentation/Domain Ontology/HTML/index.htm

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- Result_Management_System
- Classes (23)
- Object Properties (6)
- Data Properties (24)
- Annotation Properties (2)
- Individuals (23)
- Datatypes (5)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Entities (78)

- owl:Thing
- "Domain_(Requirements)"
- accessFollowingScreen
- Address
- Admin
- Availability
- Batch_year
- Branch
- Category/_Type
- Communication_interfaces
- Credit
- defineRoles
- DEO
- Design_Constraints
- Email_id
- External_Interface
- External_Interface_Requirements
- External_marks

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Result_Management_System

http://www.semanticweb.org/rohit/ontologies/2016/3/Result_Management_System

Loaded from file:/C:/Ontology%20Driven%20Software%20Developme
/Domain%20Ontology/Code/Domain.owl

Annotations (1)

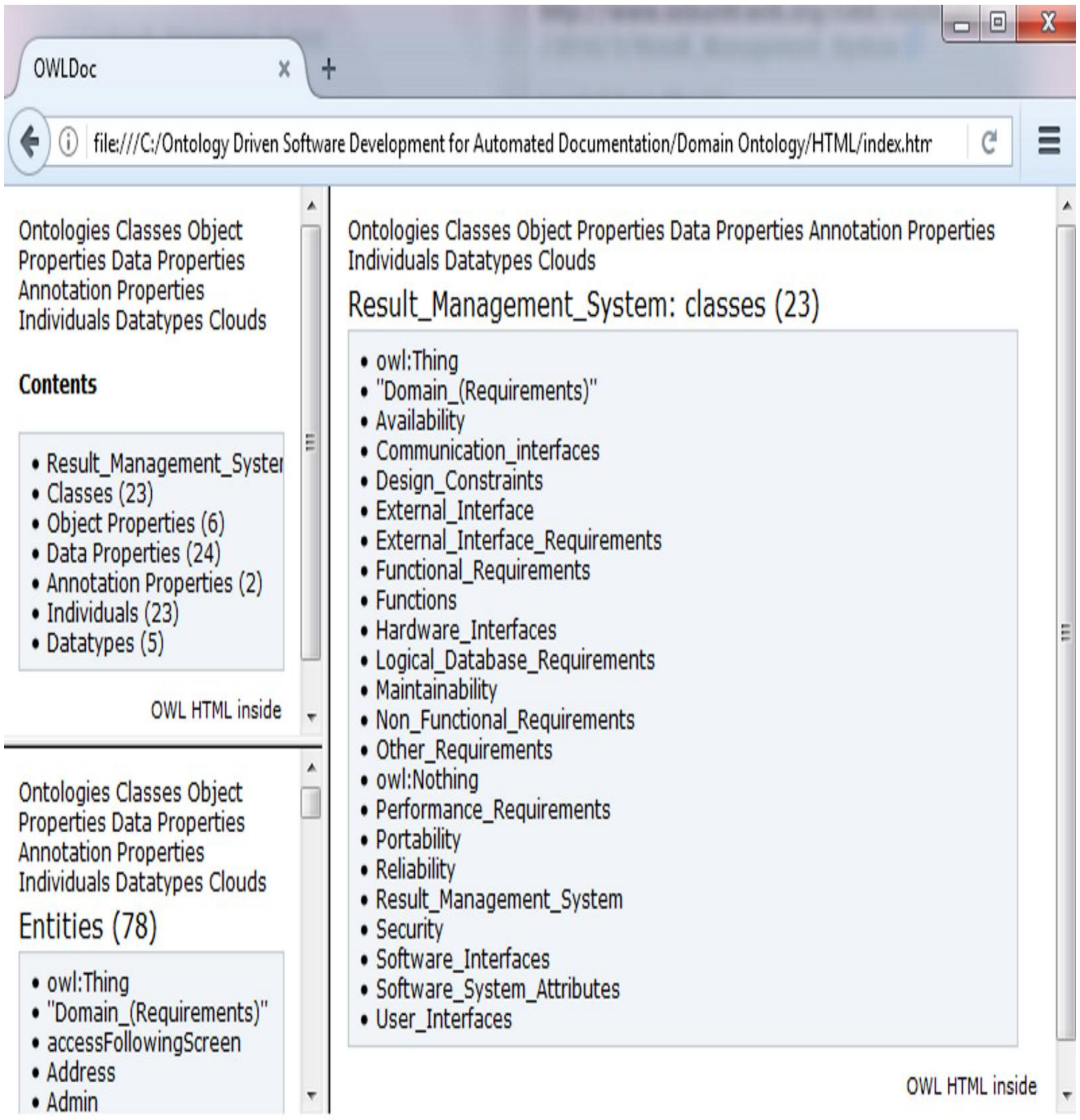
- rdfs:comment "Domain Ontology (for Result Management System)"

References

- Classes (23)
- Object Properties (6)
- Data Properties (24)
- Annotation Properties (2)
- Individuals (23)
- Datatypes (5)

OWL HTML inside

Figure 4. HTML of the index page of domain ontology


OWLDoc
file:///C:/Ontology Driven Software Development for Automated Documentation/Domain Ontology/HTML/index.htm

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- Result_Management_Syster
- Classes (23)
- Object Properties (6)
- Data Properties (24)
- Annotation Properties (2)
- Individuals (23)
- Datatypes (5)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Entities (78)

- owl:Thing
- "Domain_(Requirements)"
- accessFollowingScreen
- Address
- Admin

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Result_Management_System: classes (23)

- owl:Thing
- "Domain_(Requirements)"
- Availability
- Communication_interfaces
- Design_Constraints
- External_Interface
- External_Interface_Requirements
- Functional_Requirements
- Functions
- Hardware_Interfaces
- Logical_Database_Requirements
- Maintainability
- Non_Functional_Requirements
- Other_Requirements
- owl:Nothing
- Performance_Requirements
- Portability
- Reliability
- Result_Management_System
- Security
- Software_Interfaces
- Software_System_Attributes
- User_Interfaces

OWL HTML inside

Figure 5. HTML of domain ontology when classes hyperlink is clicked

2. For Software Requirements Specification Phase

As described this phase extends the domain ontology to create an SRS ontology which contains all the requirement specification as specified for the RMS. The SRS ontology is organized as per the IEEE guidelines and standards for SRS. This ontology has 43 classes, 4 object properties, 22 data properties, 2 annotation properties, 23 individuals, and 4 data types with a total 98 entities. Figure 6 shows a sample graph of SRS ontology with the explicitly added representation of the node and arc types used within the graph.

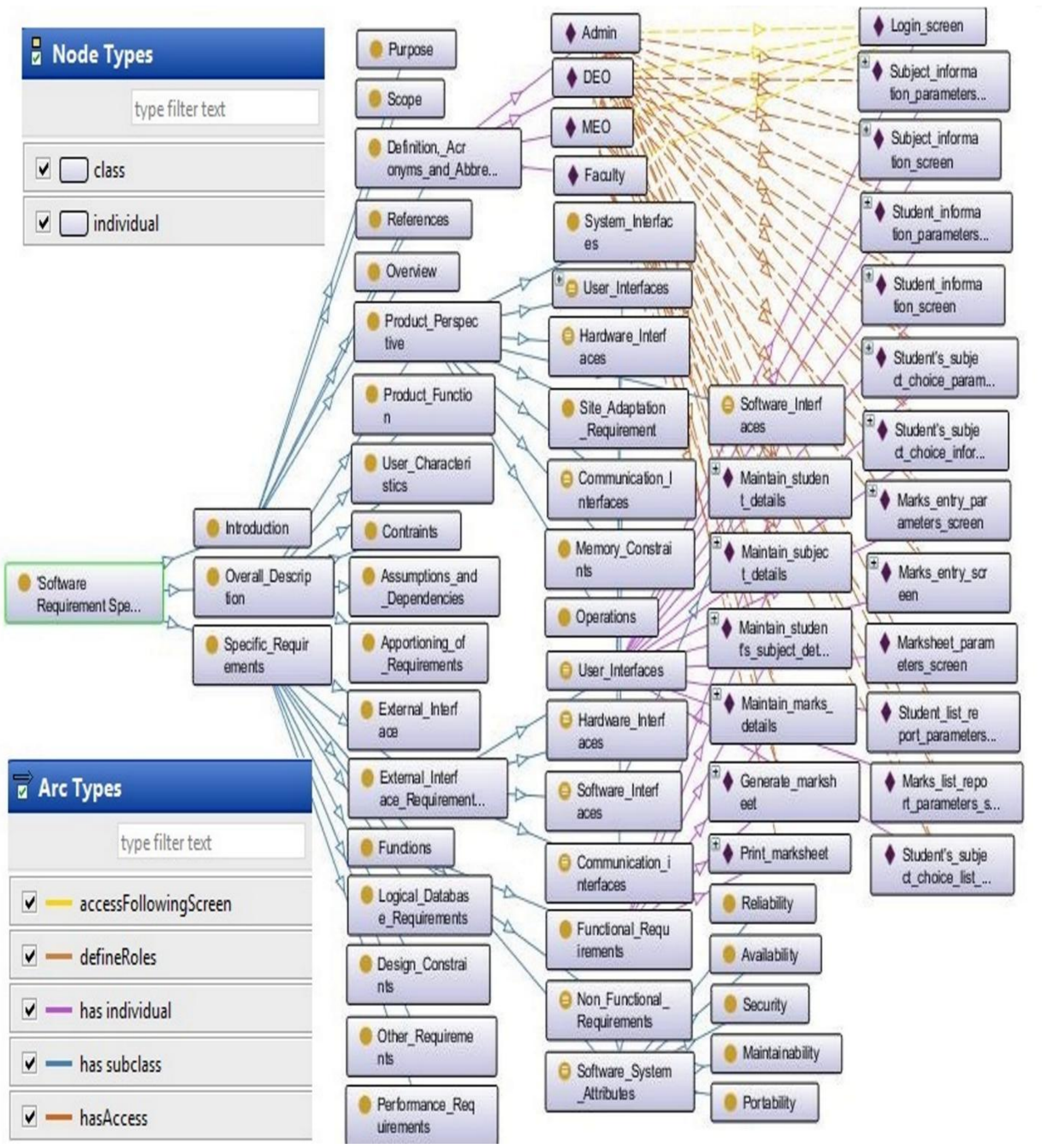
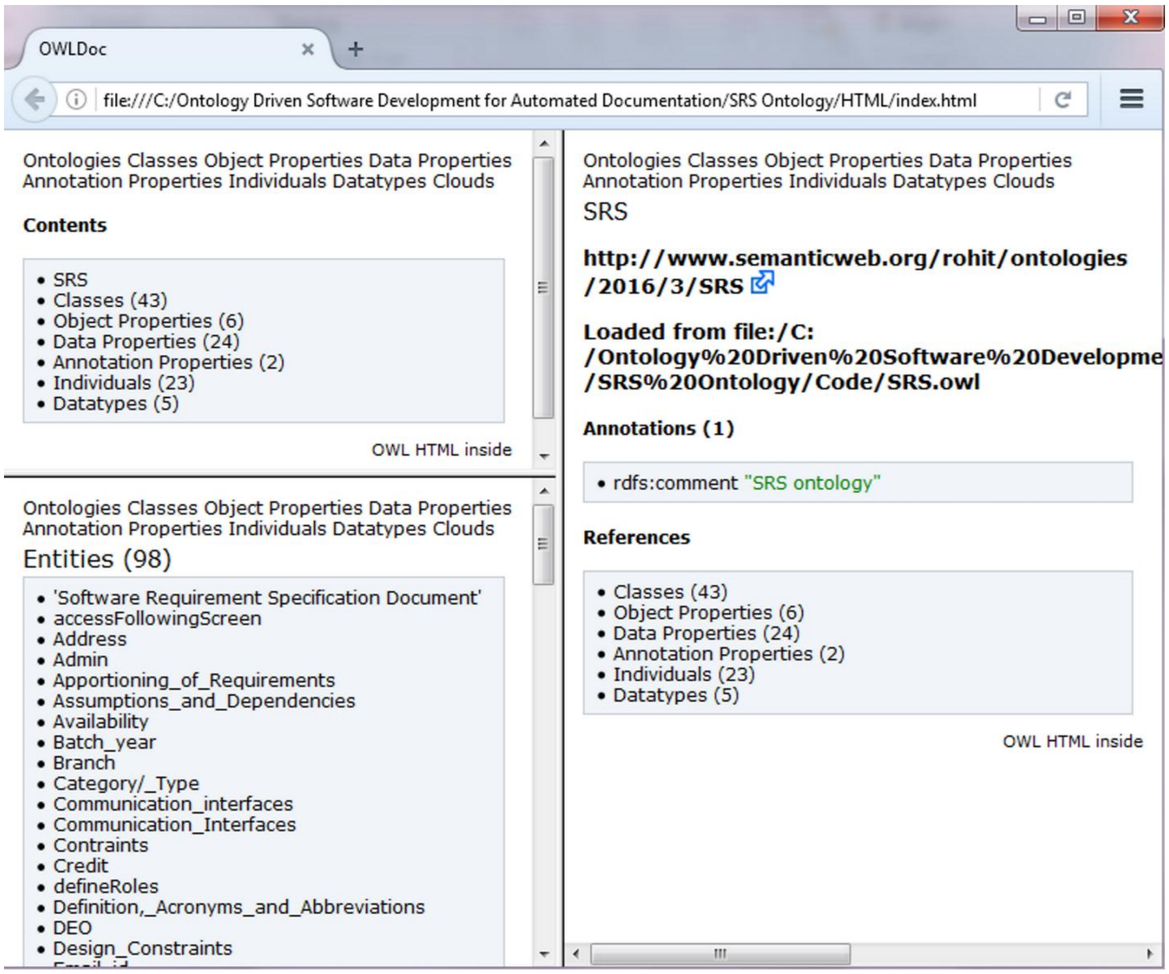


Figure 6. Graph of SRS ontology with the explicitly added representation of the node and arc types

The following Figure 7 and 8 show the HTML of the index page of SRS ontology and the HTML page of SRS ontology when its object properties hyperlink is clicked (which then gets expanded in the right-side column), respectively.


OWLDoc
file:///C:/Ontology Driven Software Development for Automated Documentation/SRS Ontology/HTML/index.html

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- SRS
- Classes (43)
- Object Properties (6)
- Data Properties (24)
- Annotation Properties (2)
- Individuals (23)
- Datatypes (5)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Entities (98)

- 'Software Requirement Specification Document'
- accessFollowingScreen
- Address
- Admin
- Apportioning_of_Requirements
- Assumptions_and_Dependencies
- Availability
- Batch_year
- Branch
- Category/_Type
- Communication_interfaces
- Communication_Interfaces
- Contraints
- Credit
- defineRoles
- Definition,_Acronyms_and_Abbreviations
- DEO
- Design_Constraints

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

SRS

http://www.semanticweb.org/rohit/ontologies/2016/3/SRS

Loaded from file:/C:/Ontology%20Driven%20Software%20Developme /SRS%20Ontology/Code/SRS.owl

Annotations (1)

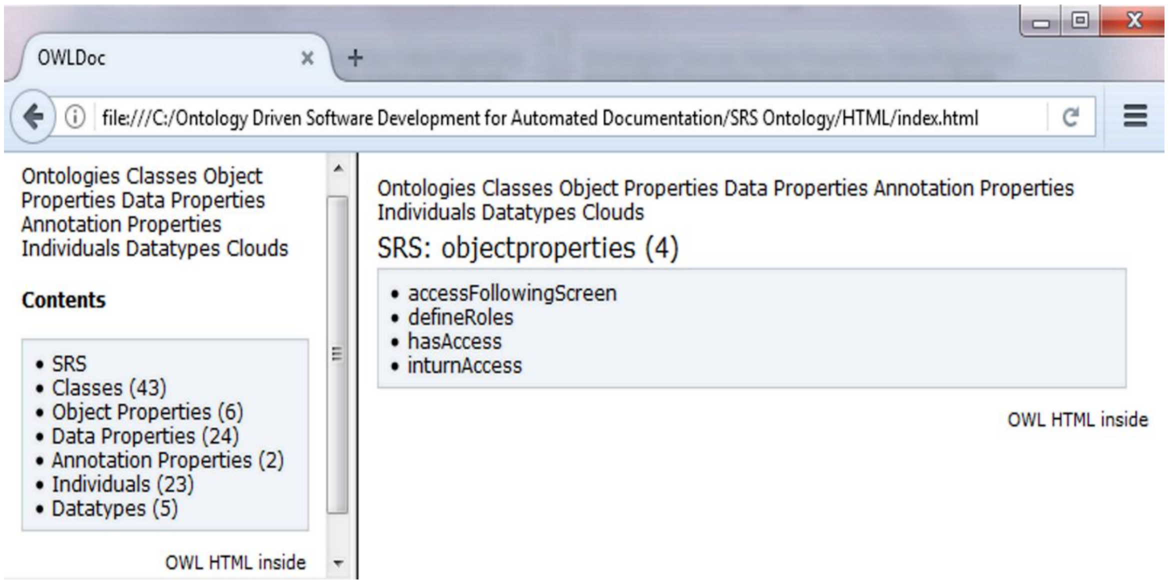
- rdfs:comment "SRS ontology"

References

- Classes (43)
- Object Properties (6)
- Data Properties (24)
- Annotation Properties (2)
- Individuals (23)
- Datatypes (5)

OWL HTML inside

Figure 7. HTML of the index page of SRS ontology


OWLDoc
file:///C:/Ontology Driven Software Development for Automated Documentation/SRS Ontology/HTML/index.html

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- SRS
- Classes (43)
- Object Properties (6)
- Data Properties (24)
- Annotation Properties (2)
- Individuals (23)
- Datatypes (5)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

SRS: objectproperties (4)

- accessFollowingScreen
- defineRoles
- hasAccess
- inturnAccess

OWL HTML inside

Figure 8. HTML of SRS ontology when object properties hyperlink is clicked

3. For Design Phase

This phase adopts the conventional design methodology, and thus UML is used to produce the Software Design Document (SDD). Sample snapshots of UML diagrams generated for RMS are shown below. Figure 9 explains the sequence diagram of basic login flow where a user tries to login the system. A user may have different roles such as Admin, Data Entry Operator (DEO), Marks Entry Operator (MEO), etc. Basic login flow comprises a sequence where the user provides username and password to login the system and accordingly sequence is followed which is as shown in the figure below.

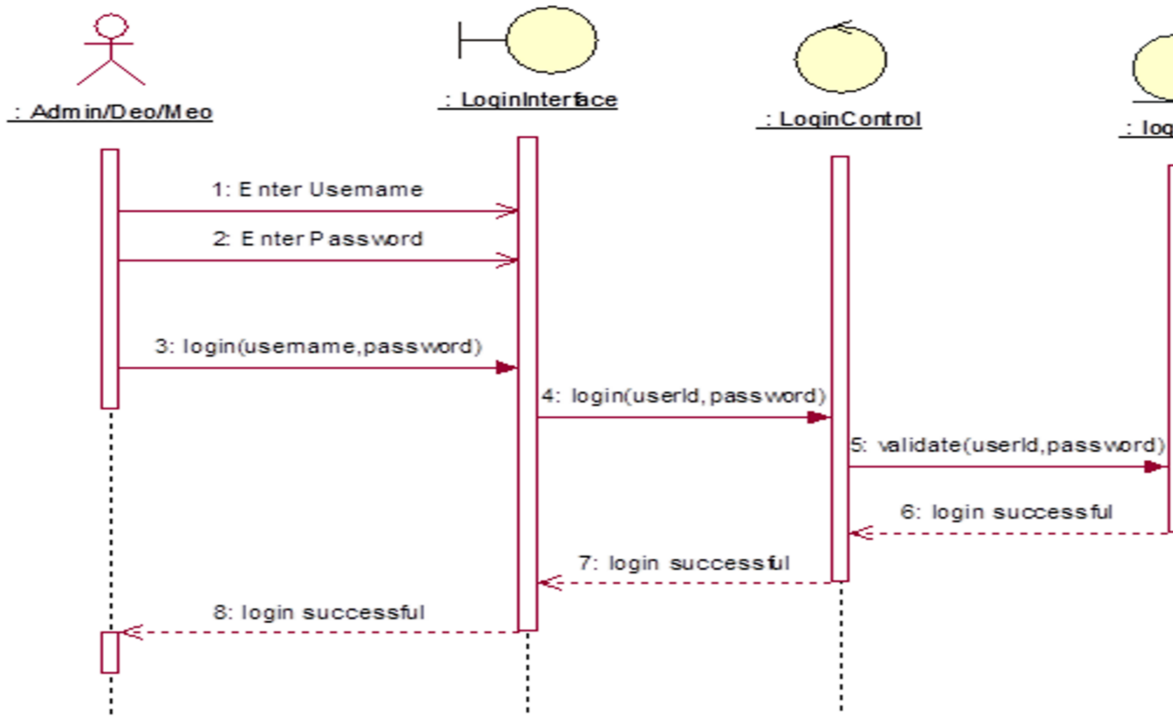


Figure 9. UML sequence diagram for basic login flow

Figure 10 describes the activity diagram for generation of mark sheet. It represents the flow of control from activity to activity for generation of mark sheet right from beginning to end, i.e., from initializing to exit as shown in the figure below.

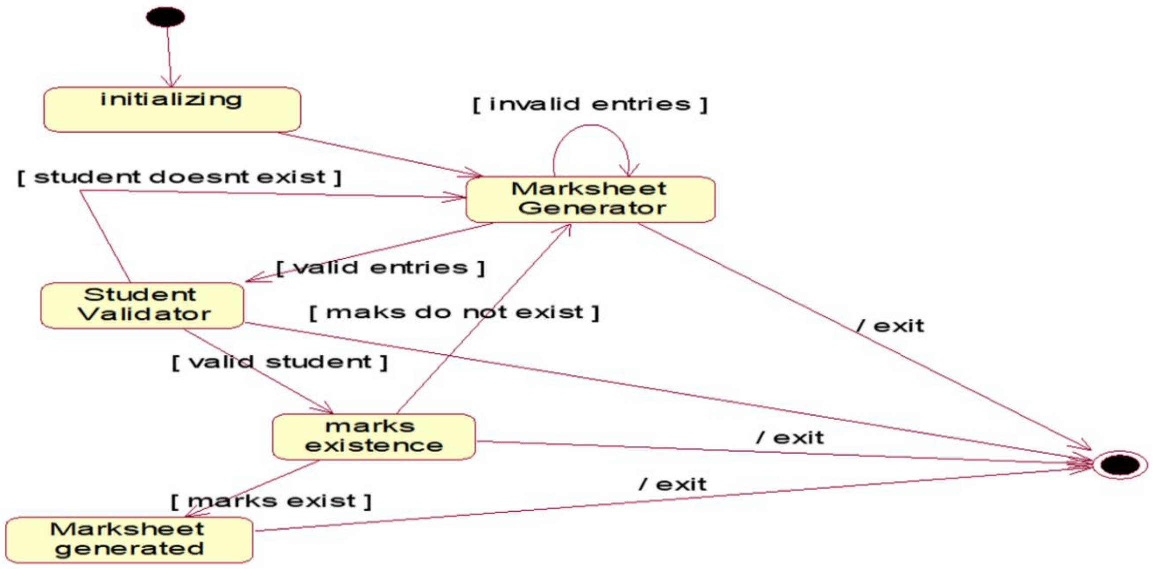


Figure 10. UML activity diagram for generation of mark sheet

4. For Source Code Phase

RMS is developed using an Object-Oriented Programming language. It uses Visual Basic programming language and Hypertext Preprocessor (PHP). Figure 11 and 12 depict the snapshot of Graphical User Interface (GUI) of RMS student details page and mark sheet generator home page.

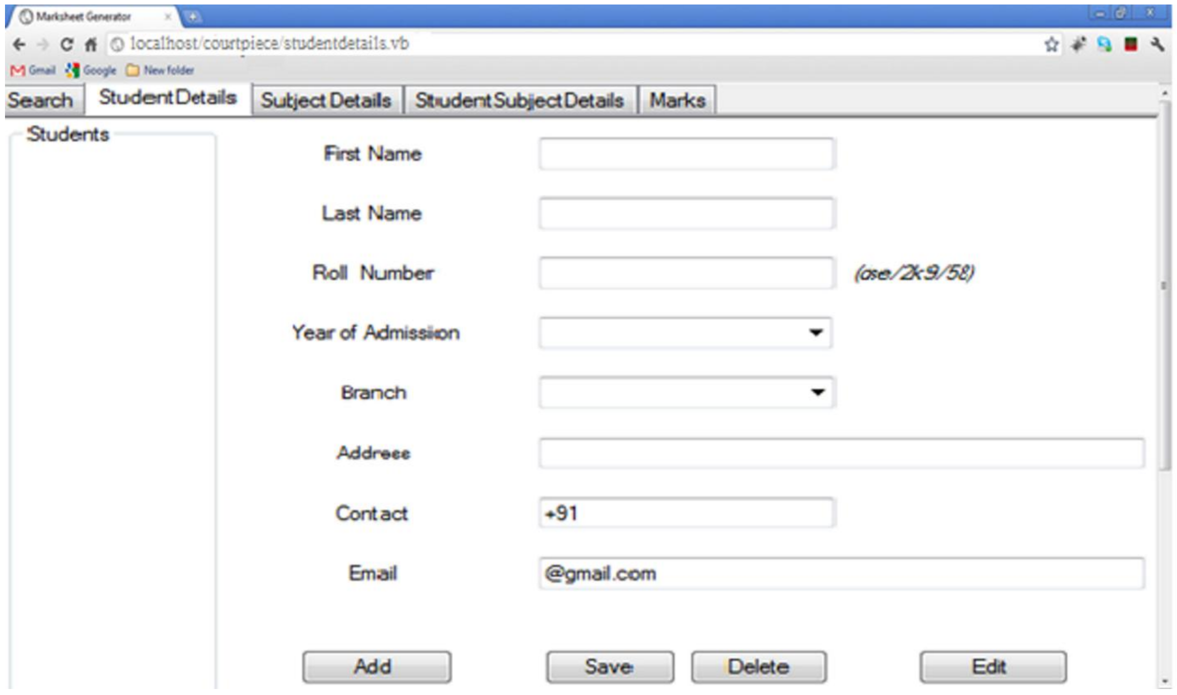


Figure 11. RMS student details page

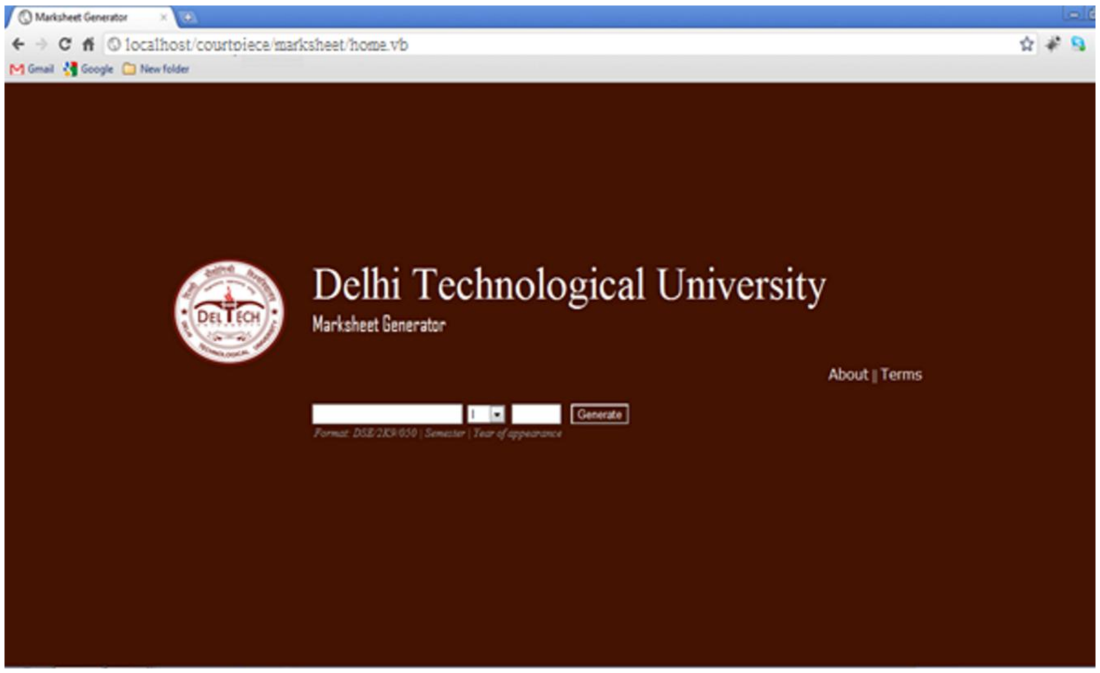

Marksheet Generator
localhost/courtpiece/marksheet/home.vb
Gmail
Google
New folder
Delhi Technological University
Marksheet Generator
About | Terms
Generate
Format: DSE/2K9/050 | Semester | Year of appearance

Figure 12. RMS mark sheet generator home page

5. For Source Code Listing Ontology Phase

In this phase, we list out all the forms, variables, and functions along with their functionalities that are used for implementation or coding phase. This ontology has 10 classes, 0 object properties, 0 data properties, 1 annotation properties, 82 individuals, and 1 data type with a total of 94 entities. Figure 13 shows the graph of source code listing ontology with the explicitly added representation of the node and arc types used within the graph.

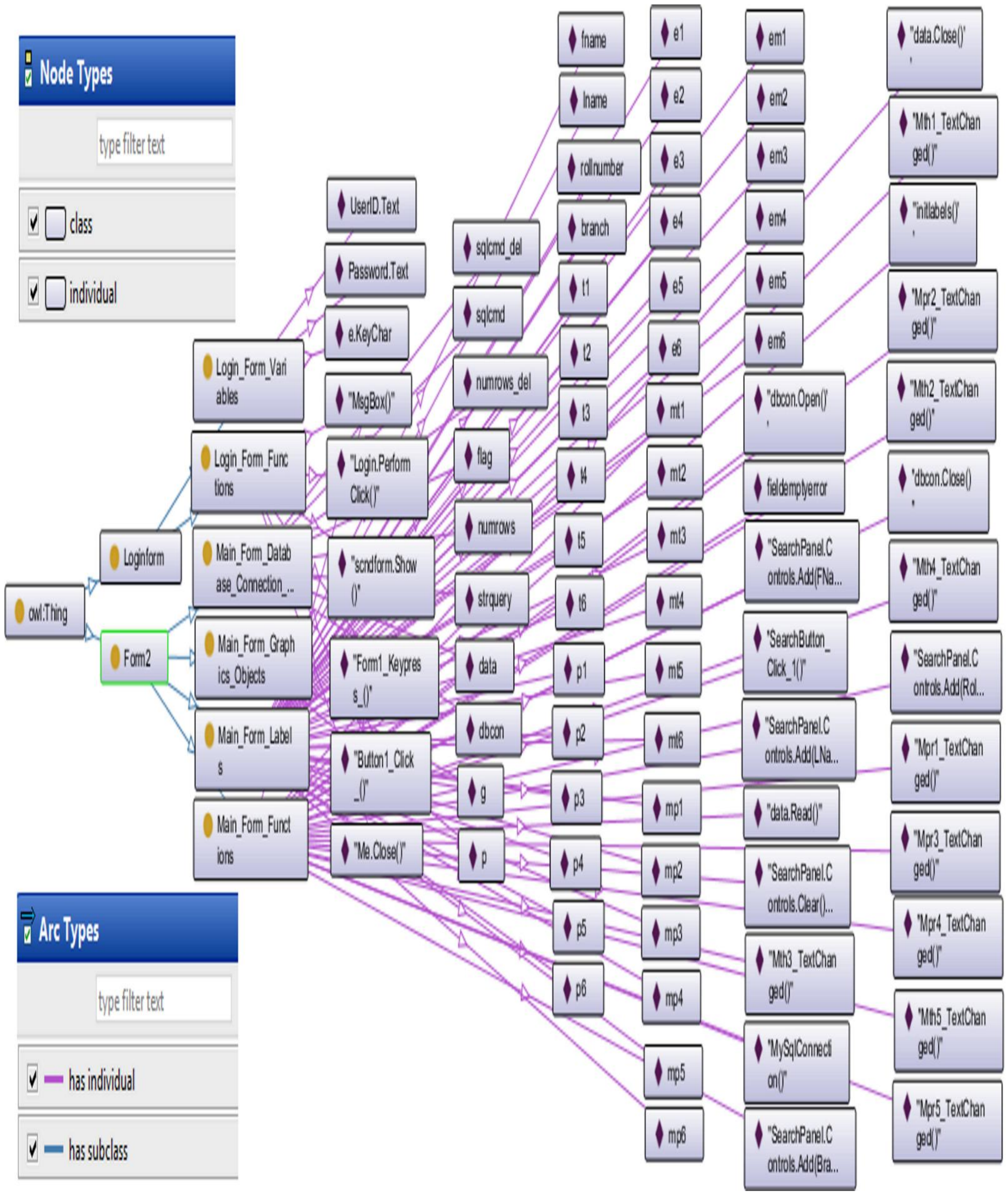
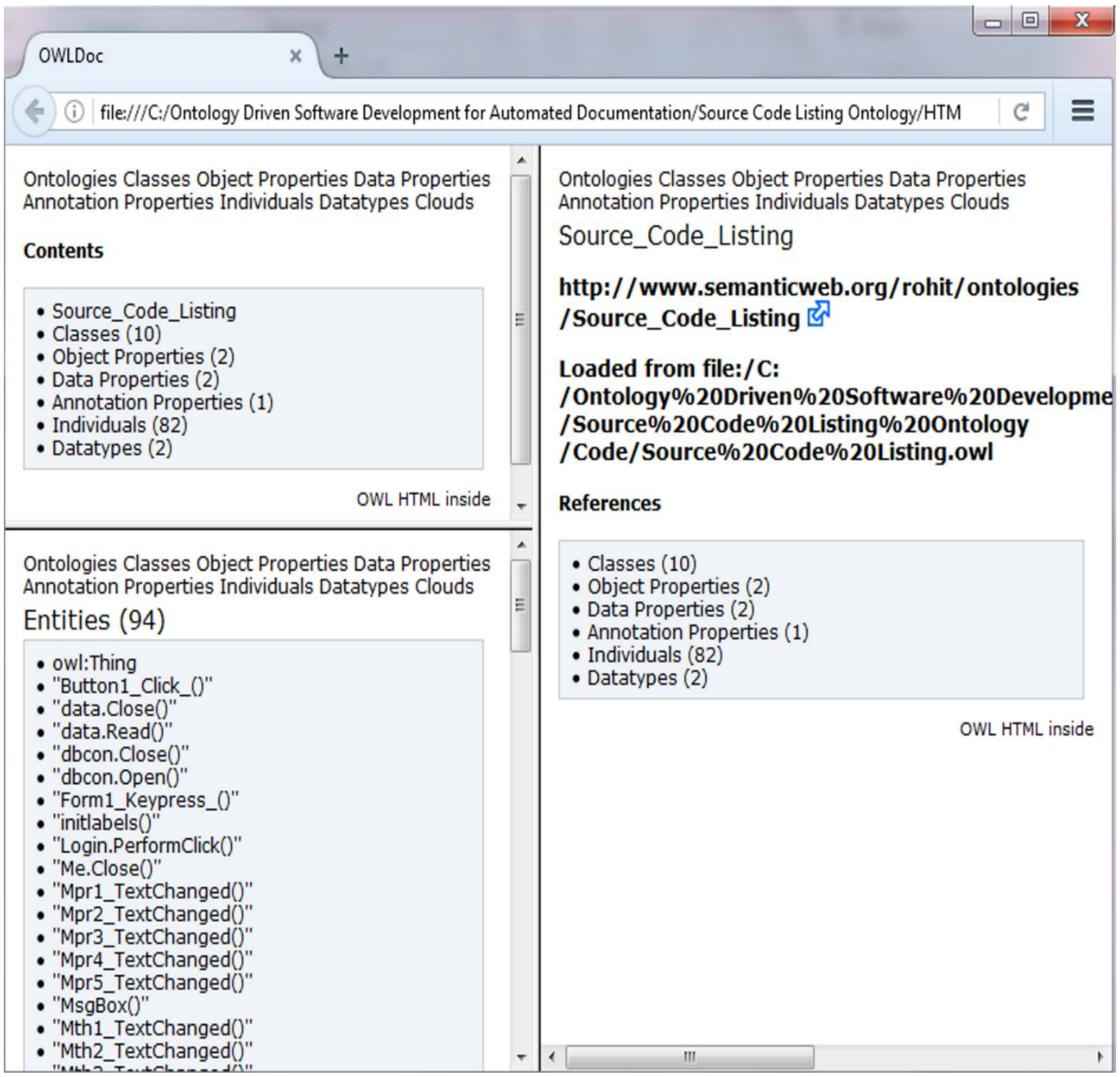


Figure 13. Graph of source code listing ontology with the explicitly added representation of the node and arc types

Figure 14 shows the HTML of the index page of source code listing ontology.


OWLDoc

file:///C:/Ontology Driven Software Development for Automated Documentation/Source Code Listing Ontology/HTM

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- Source_Code_Listing
- Classes (10)
- Object Properties (2)
- Data Properties (2)
- Annotation Properties (1)
- Individuals (82)
- Datatypes (2)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Entities (94)

- owl:Thing
- "Button1_Click_()"
- "data.Close()"
- "data.Read()"
- "dbcon.Close()"
- "dbcon.Open()"
- "Form1_Keypress_()"
- "initlabels()"
- "Login.PerformClick()"
- "Me.Close()"
- "Mpr1_TextChanged()"
- "Mpr2_TextChanged()"
- "Mpr3_TextChanged()"
- "Mpr4_TextChanged()"
- "Mpr5_TextChanged()"
- "MsgBox()"
- "Mth1_TextChanged()"
- "Mth2_TextChanged()"

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Source_Code_Listing

http://www.semanticweb.org/rohit/ontologies/Source_Code_Listing

Loaded from file:/C:/Ontology%20Driven%20Software%20Developme/Source%20Code%20Listing%20Ontology/Code/Source%20Code%20Listing.owl

References

- Classes (10)
- Object Properties (2)
- Data Properties (2)
- Annotation Properties (1)
- Individuals (82)
- Datatypes (2)

OWL HTML inside

Figure 14. HTML of the index page of source code listing ontology

6. For Test Cases Ontology Phase

For each scenario, test case ontologies are built in this phase. For example, login test cases ontology, maintain student’s details test cases ontology, maintain subject’s details test cases ontology, maintain student's subjects test cases ontology, generate mark sheet test cases ontology, and print mark sheet test cases ontology. However, to reduce the redundancy of no. of graphs and HTML pages of various scenarios, we included the snapshots of login test cases and maintain student’s details test cases ontologies only.

Login test cases ontology has 7 classes, 2 object properties, 1 data properties, 1 annotation properties, 7 individuals, and 3 data type with a total of 21 entities. Figure 15 shows the graph of login test cases ontology with super embedded node and arc types.

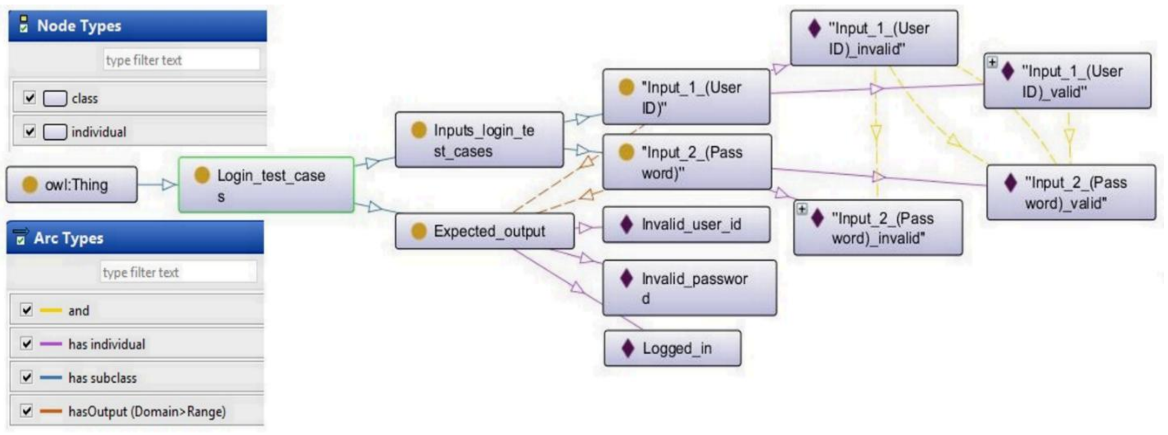
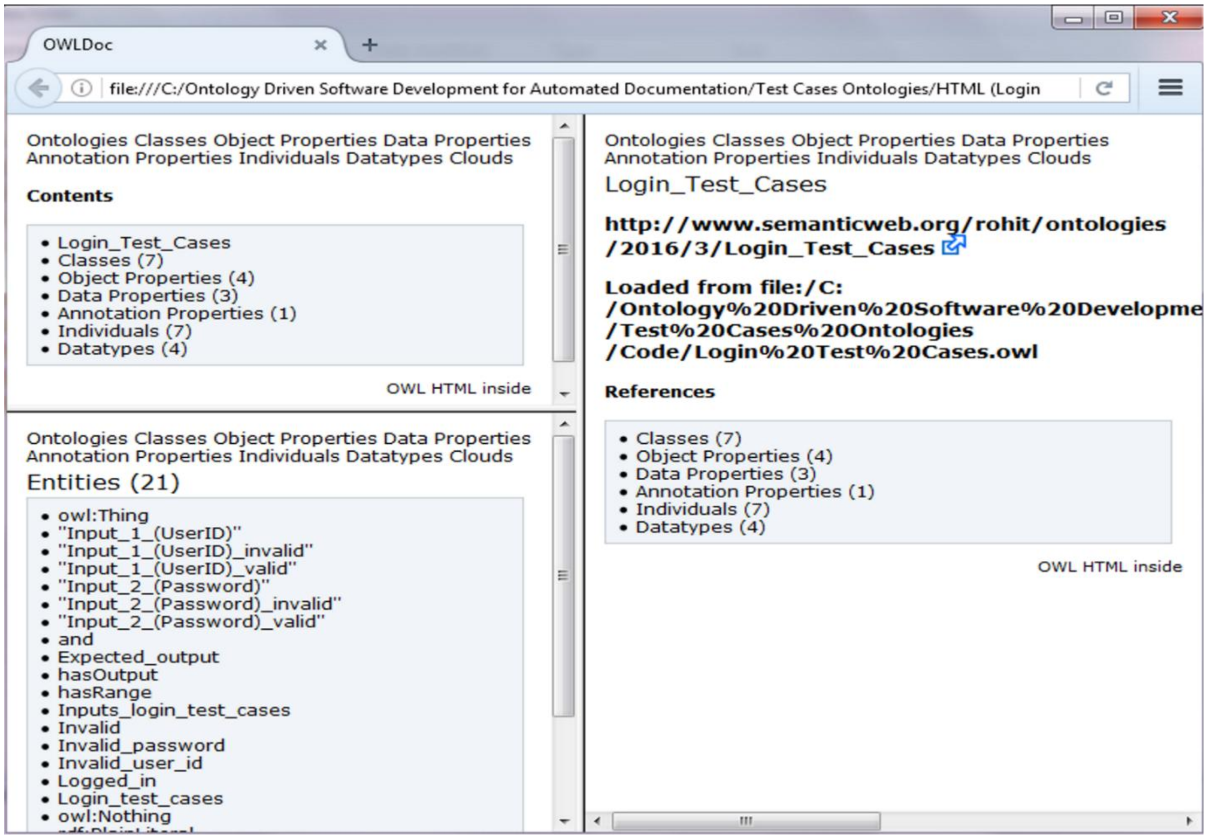


Figure 15. Graph of login test cases ontology with super embedded node and arc types

Figure 16 shows the HTML of the index page of login test cases ontology.



OWLDoc
file:///C:/Ontology Driven Software Development for Automated Documentation/Test Cases Ontologies/HTML (Login

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- Login_Test_Cases
- Classes (7)
- Object Properties (4)
- Data Properties (3)
- Annotation Properties (1)
- Individuals (7)
- Datatypes (4)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Entities (21)

- owl:Thing
- "Input_1_(UserID)"
- "Input_1_(UserID)_invalid"
- "Input_1_(UserID)_valid"
- "Input_2_(Password)"
- "Input_2_(Password)_invalid"
- "Input_2_(Password)_valid"
- and
- Expected_output
- hasOutput
- hasRange
- Inputs_login_test_cases
- Invalid
- Invalid_password
- Invalid_user_id
- Logged_in
- Login_test_cases
- owl:Nothing

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Login_Test_Cases

http://www.semanticweb.org/rohit/ontologies/2016/3/Login_Test_Cases

Loaded from file:/C:/Ontology%20Driven%20Software%20Developme/Test%20Cases%20Ontologies/Code/Login%20Test%20Cases.owl

References

- Classes (7)
- Object Properties (4)
- Data Properties (3)
- Annotation Properties (1)
- Individuals (7)
- Datatypes (4)

OWL HTML inside

Figure 16. HTML of the index page of login test cases ontology

Maintain student's details test cases ontology has 8 classes, 2 object properties, 1 data properties, 1 annotation properties, 15 individuals, and 3 data type with a total of 30 entities. Figure 17 shows the graph of maintain student's details test cases ontology with super embedded node and arc types.

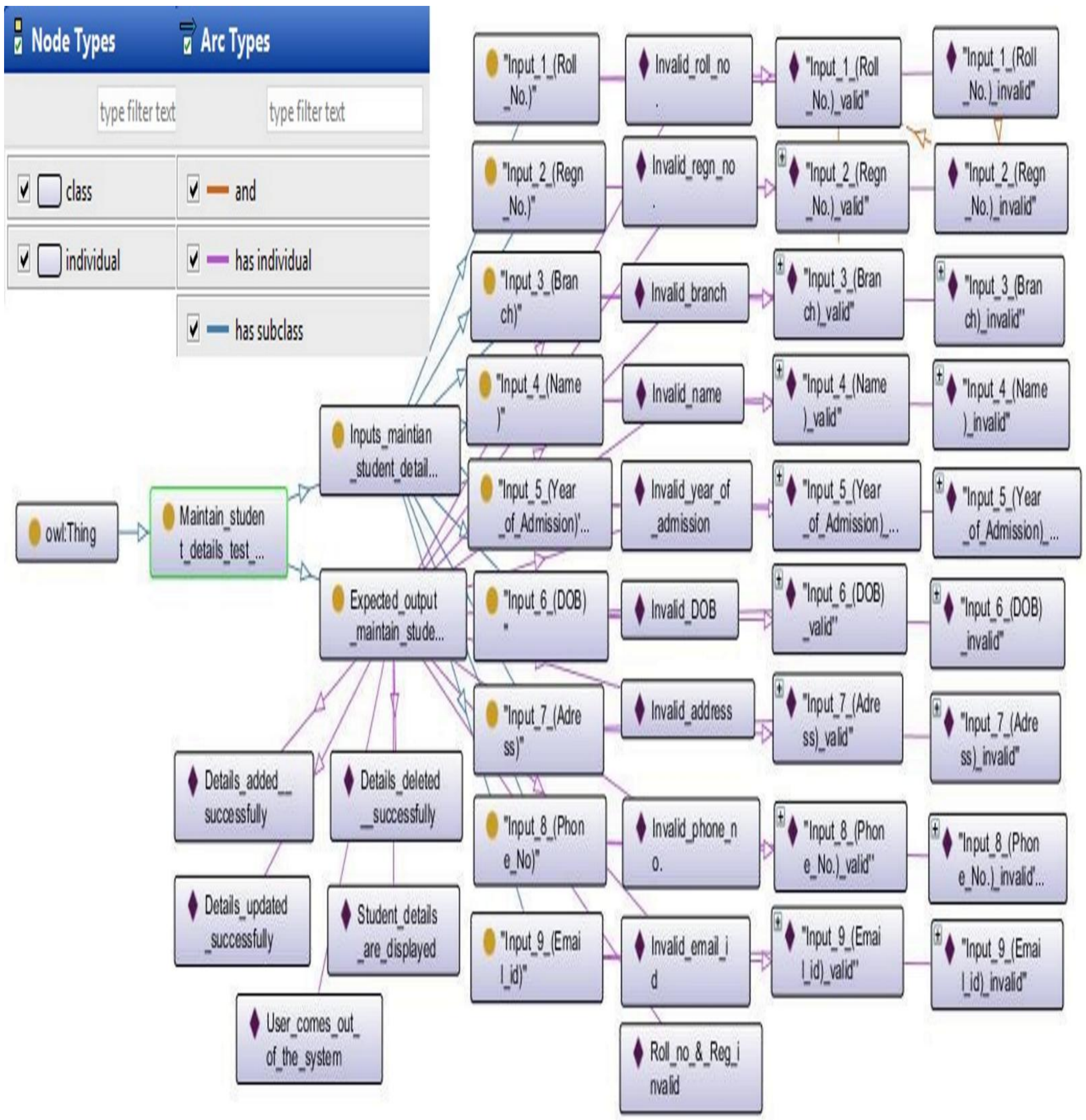
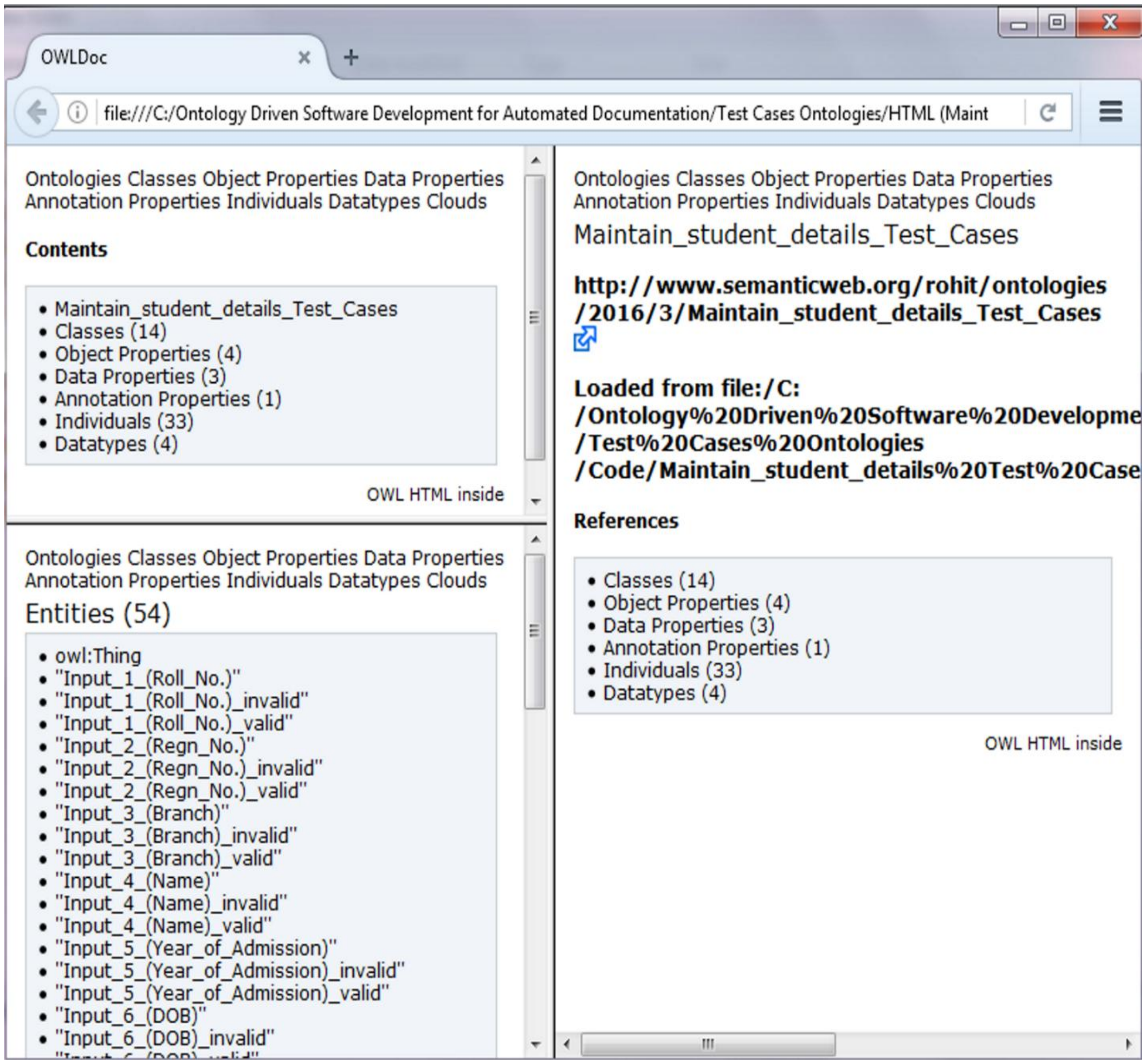

Node Types
Arc Types
type filter text
type filter text
class
and
individual
has individual
has subclass
owl:Thing
Maintain_student_details_test_...
Inputs_maintian_student_detail...
Expected_output_maintain_stude...
"Input_1_(Roll No.)"
"Input_2_(Regn No.)"
"Input_3_(Branch)"
"Input_4_(Name)"
"Input_5_(Year_of_Admission)'...
"Input_6_(DOB)"
"Input_7_(Adress)"
"Input_8_(Phone_No)"
"Input_9_(Email_id)"
Invalid_roll_no.
Invalid_regn_no.
Invalid_branch
Invalid_name
Invalid_year_of_admission
Invalid_DOB
Invalid_address
Invalid_phone_no.
Invalid_email_id
Roll_no_&_Reg_invalid
"Input_1_(Roll_No.)_valid"
"Input_2_(Regn_No.)_valid"
"Input_3_(Branch)_valid"
"Input_4_(Name)_valid"
"Input_5_(Year_of_Admission)_...
"Input_6_(DOB)_valid"
"Input_7_(Adress)_valid"
"Input_8_(Phone_No.)_valid"
"Input_9_(Email_id)_valid"
"Input_1_(Roll_No.)_invalid"
"Input_2_(Regn_No.)_invalid"
"Input_3_(Branch)_invalid"
"Input_4_(Name)_invalid"
"Input_5_(Year_of_Admission)_...
"Input_6_(DOB)_invalid"
"Input_7_(Adress)_invalid"
"Input_8_(Phone_No.)_invalid'...
"Input_9_(Email_id)_invalid"
Details_added_successfully
Details_deleted_successfully
Details_updated_successfully
Student_details_are_displayed
User_comes_out_of_the_system

Figure 17. Graph of maintain student's details test cases ontology with super embedded node and arc types

Figure 18 shows the HTML of the index page of maintain student's details test cases ontology.


OWLDoc
file:///C:/Ontology Driven Software Development for Automated Documentation/Test Cases Ontologies/HTML (Maint

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Contents

- Maintain_student_details_Test_Cases
- Classes (14)
- Object Properties (4)
- Data Properties (3)
- Annotation Properties (1)
- Individuals (33)
- Datatypes (4)

OWL HTML inside

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Entities (54)

- owl:Thing
- "Input_1_(Roll_No.)"
- "Input_1_(Roll_No.)_invalid"
- "Input_1_(Roll_No.)_valid"
- "Input_2_(Regn_No.)"
- "Input_2_(Regn_No.)_invalid"
- "Input_2_(Regn_No.)_valid"
- "Input_3_(Branch)"
- "Input_3_(Branch)_invalid"
- "Input_3_(Branch)_valid"
- "Input_4_(Name)"
- "Input_4_(Name)_invalid"
- "Input_4_(Name)_valid"
- "Input_5_(Year_of_Admission)"
- "Input_5_(Year_of_Admission)_invalid"
- "Input_5_(Year_of_Admission)_valid"
- "Input_6_(DOB)"
- "Input_6_(DOB)_invalid"

Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Maintain_student_details_Test_Cases

http://www.semanticweb.org/rohit/ontologies/2016/3/Maintain_student_details_Test_Cases

Loaded from file:/C:/Ontology%20Driven%20Software%20Developme/Test%20Cases%20Ontologies/Code/Maintain_student_details%20Test%20Case

References

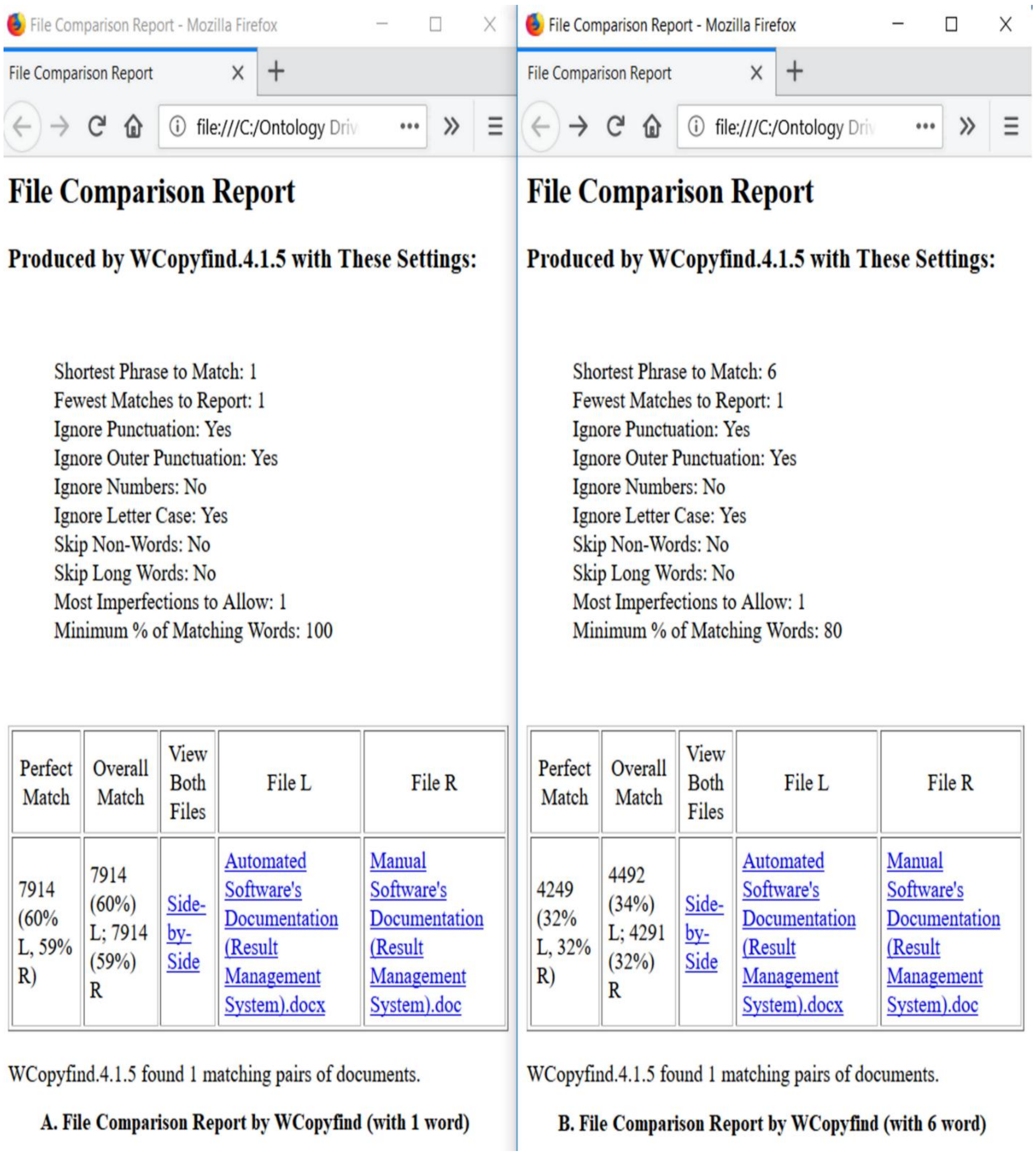
- Classes (14)
- Object Properties (4)
- Data Properties (3)
- Annotation Properties (1)
- Individuals (33)
- Datatypes (4)

OWL HTML inside

Figure 18. HTML of the index page of maintain student's details test cases ontology

Result and Analysis

All the ontologies are combined to generate consolidated text document which accomplishes the primary goal of automated documentation. To evaluate and verify the proposed framework for automation, we compare the automated documentation to the existing manual documentation for the RMS software. The similarity index between the two documentations is calculated using the software WCopyfind version 4.1.5, an open source software for comparing two textual documents. It generates an output report with similarities in words and phrases within the two documents (Bloomfield, n.d.). The percentage of similarity index is indicative of the proportion of automation achieved. The result of the comparison is shown along with the adjusted settings of WCopyfind in Figure 19.



File Comparison Report

Produced by WCopyfind.4.1.5 with These Settings:

Shortest Phrase to Match: 1
Fewest Matches to Report: 1
Ignore Punctuation: Yes
Ignore Outer Punctuation: Yes
Ignore Numbers: No
Ignore Letter Case: Yes
Skip Non-Words: No
Skip Long Words: No
Most Imperfections to Allow: 1
Minimum % of Matching Words: 100

Perfect Match	Overall Match	View Both Files	File L	File R
7914 (60% L, 59% R)	7914 (60%) L; 7914 (59%) R	Side-by-Side	Automated Software's Documentation (Result Management System).docx	Manual Software's Documentation (Result Management System).doc

WCopyfind.4.1.5 found 1 matching pairs of documents.

A. File Comparison Report by WCopyfind (with 1 word)

File Comparison Report

Produced by WCopyfind.4.1.5 with These Settings:

Shortest Phrase to Match: 6
Fewest Matches to Report: 1
Ignore Punctuation: Yes
Ignore Outer Punctuation: Yes
Ignore Numbers: No
Ignore Letter Case: Yes
Skip Non-Words: No
Skip Long Words: No
Most Imperfections to Allow: 1
Minimum % of Matching Words: 80

Perfect Match	Overall Match	View Both Files	File L	File R
4249 (32% L, 32% R)	4492 (34%) L; 4291 (32%) R	Side-by-Side	Automated Software's Documentation (Result Management System).docx	Manual Software's Documentation (Result Management System).doc

WCopyfind.4.1.5 found 1 matching pairs of documents.

B. File Comparison Report by WCopyfind (with 6 word)

Figure 19. File Comparison Report by WCopyfind (with 01 and 06 words)

From figure 19, part A, it is clear that 60 percent automation is achieved with the above-adjusted settings of WCopyfind. Increasing the shortest phrase to match limit to 06 words provides us around 32 percent automation which is also clearly depicted in part B when the phrase limit is changed to 06 words and keeping other settings as intact as in part A. Table 2 shows how the percentage of automation achieved varies when the different settings of WCopyfind are applied.

Table 2. Automation achieved when the different settings of WCopyfind are applied

WCopyfind Settings Properties	Setting 1	Setting 2	Setting 3	Setting 4	Settings 5
Shortest Phrase to Match	1	6	12	18	24
Fewest Matches to Report	1	1	1	1	1
Ignore Punctuation	Yes	Yes	Yes	Yes	Yes
Ignore Outer Punctuation	Yes	Yes	Yes	Yes	Yes
Ignore Numbers	No	No	No	No	No
Ignore Letter Case	Yes	Yes	Yes	Yes	Yes
Skip Non-Words	No	No	No	No	No
Skip Long Words	No	No	No	No	No
Most Imperfections to Allow	1	1	1	1	1
Minimum % of Matching Words	80	80	80	80	80
Overall Match (Automation)	60%	32%	30%	29%	28%

Figure 20 summarizes the tabular report given in table 2 in bar chart form which clearly shows how the percentage of automation achieved varies when different settings of WCopyfind are applied.

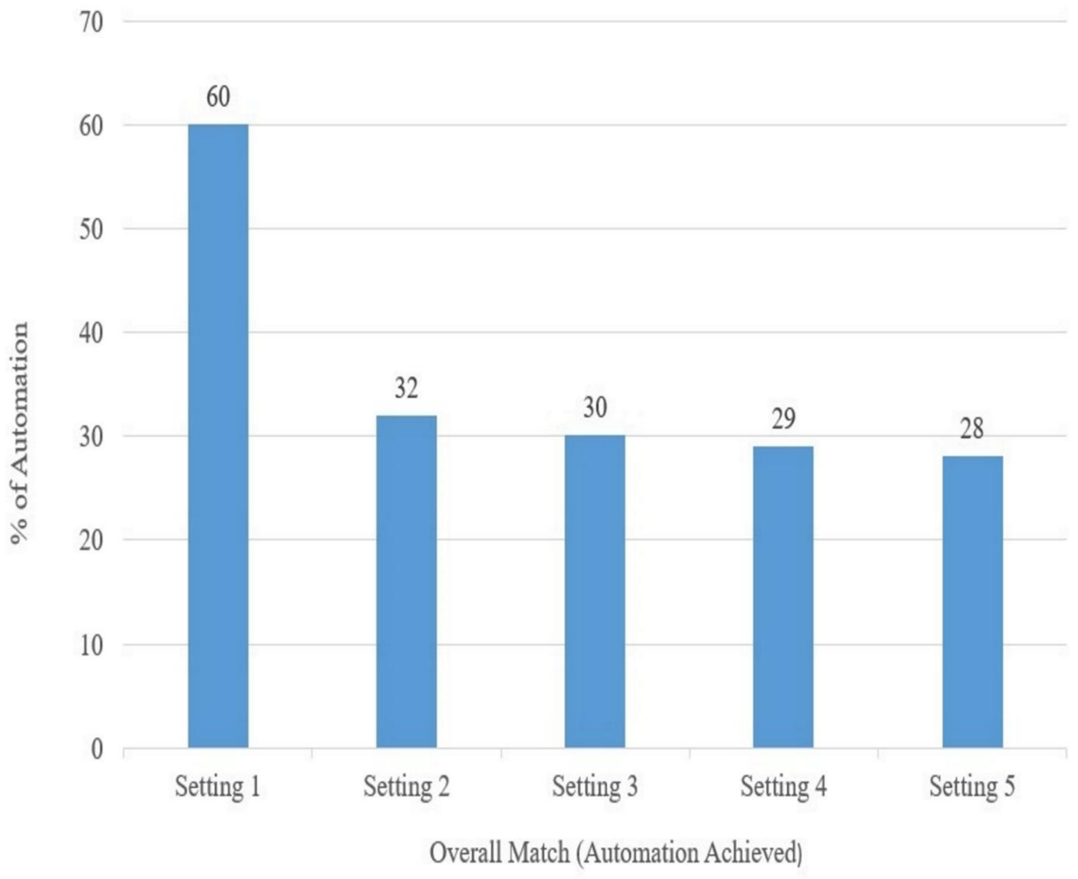


Figure 20. Bar chart showing how the percentage of automation achieved varies when different settings of WCopyfind are applied

From figure 20, it is visible that there is a sharp decline in the automation achieved, i.e., from 60 percent to 32 percent when we go from setting 1 to setting 2. The reason for this sharp fall is because shortest phrase to match changes from 01 to 06 in setting 1 to setting 2. However, after that, there is no major decline in automation achieved even in setting 5 where shortest phrase to match is 24 words. Therefore, it is clear that our approach for automated documentation provided a stable and reliable solution even when shortest phrase to match is increased significantly.

Conclusion and Future Scope

Here in this paper, we presented ontology driven software development approach for automated documentation. We displayed the framework for our approach and exhibited its implementation. Moreover, comparison of our approach of automated documentation with the manual one shows 60 percent automation with 01word of shortest phrase to match and 32 percent in case of 06 words of shortest phrase to match. Here we would like to mention that percentage of automation may vary from one case study to another. Also, Wcopyfind software provides similarity index based on similar text and phrases found between two documents. It does not show any similarity when two statements are written differently, yet, conveying the same message. Therefore, the percentage of similarity index is just an indicator of the proportion of automation achieved, nevertheless, actual automation may be even more. Therefore, our effort for automated documentation provides very promising results that the proposed approach is definitely better when it is compared to the manual one. Future work in this area can include further optimization of automated documentation to achieve a higher percentage of automation.

References

American Library Association, Office of Intellectual Freedom. (2013). *Banned and challenged books.* Retrieved September 15, 2018, from http://www.ala.org/advocacy/banned/bannedbooksweek

American Library Association Council. (1939). *Library bill of rights.* Retrieved September 15, 2018, from http://www.ala.org/advocacy/intfreedom/librarybill

Anunobi, C. V., Ifeyinwa, B., & Okoye, I. B. (2008). The role of academic libraries in universal access to print and electronic resources in the developing countries. *Library Philosophy and Practice*, May, 1-5. Retrieved September 15, 2018, from http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1194&context=libphilprac

Foerstel, H. N. (2002). *Banned in the U.S.A.: A reference guide to book censorship in schools and*

Jenkins, C. A. (2008). Book challenges, challenging books, and young readers: The research picture. *Language Arts*, 85(3), 228-236.

Knight, J., & King, C. (2010). A question of accessibility: Exploring barriers to the freedom of Information Flow. *Journal of Library and Information Science*, 39(2), 93-101.

Knox, E.J.M. (2015). *Book Banning in 21st Century America*. Lanham, MD: Rowman & Littlefield.

Liu, Z. (2005). Reading behaviour in the digital environment: Changes in reading behaviour over the past 10 years. *Journal of Documentation*, 61(6), 700-712.

Bhatia, M. P. S., Beniwal, R., & Kumar, A. (2014). An ontology based framework for automatic detection and updation of requirement specifications. In *Contemporary Computing and Informatics (IC3I), 2014 International Conference on* (pp. 238-242). IEEE.

Bhatia, M. P. S., Kumar, A., & Beniwal, R. (2015). Ontology based framework for automatic software's documentation. In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on* (pp. 421-424). IEEE.

Bhatia, M. P. S., Kumar, A., & Beniwal, R. (2016a). Ontologies for software engineering: Past, present and future. *Indian Journal of Science and Technology*, 9(9).

Bhatia, M. P. S., Kumar, A., & Beniwal, R. (2016b). Ontology based framework for reverse engineering of conventional softwares. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on* (pp. 3645-3648). IEEE.

Bhatia, M. P. S., Kumar, A., & Beniwal, R. (2016c). SWOT Analysis of Ontology Driven Software Engineering. *Indian Journal of Science and Technology*, 9(38).

Bloomfield, L. (n.d.). *WCopyfind*. Retrieved September 15, 2018, from http://plagiarism.bloomfieldmedia.com/wordpress/software/wcopyfind/

Cockburn, A. (2002). *Agile software development*. Vol. 177. Boston: Addison-Wesley.

de Graaf, K. A. (2011). Annotating software documentation in semantic wikis. In *Proceedings of the fourth workshop on Exploiting semantic annotations in information retrieval* (pp. 5-6). ACM.

de Graaf, K. A., Liang, P., Tang, A., van Hage, W. R., & van Vliet, H. (2014). An exploratory study on ontology engineering for software architecture documentation. *Computers in Industry*, 65(7), 1053-1064.

Ding, L., Kolari, P., Ding, Z., & Avancha, S. (2007). Using ontologies in the semantic web: A survey. In *Ontologies* (pp. 79-113). Springer US.

Drummond, N., Horridge, M., & Redmond, T. (2012). *OWLDoc*. Retrieved September 15, 2018, from https://protegewiki.stanford.edu/wiki/OWLDoc

Drummond, N. (n.d.) *OWLDoc*. Retrieved from http://code.google.com/p/co-ode-owl-plugins/wiki/OWLDoc

Falconer, S. (2013). *OntoGraf*. Retrieved September 15, 2018, from https://protegewiki.stanford.edu/wiki/OntoGraf

Gašević, D., Kaviani, N., & Milanović, M. (2009). Ontologies and software engineering. In *Handbook on Ontologies* (pp. 593-615). Springer, Berlin, Heidelberg.

IEEE Computer Society. (1998a). IEEE Standard for Software Project Management Plans. *IEEE STD 1058-1998*.

IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998b). IEEE recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers.

IEEE Standards Coordinating Committee. (1990). IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos. *CA: IEEE Computer Society*, 169.

Koukias, A., Nadoveza, D., & Kiritsis, D. (2015). An ontology–based approach for modelling technical documentation towards ensuring asset optimisation. International *Journal of Product Lifecycle Management*, 8(1), 24-45.

Koukias, A., & Kiritsis, D. (2015). Rule-based mechanism to optimize asset management using a technical documentation ontology. *IFAC-Papers OnLine*, 48(3), 1001-1006.

Kruchten, P., Capilla, R., & Dueñas, J. C. (2009). The decision view's role in software architecture practice. *IEEE Software*, 26(2), 36-42.

Kumar, A., & Gupta, D. (2018). Paradigm shift from conventional software quality models to web based quality models. *International Journal of Hybrid Intelligent Systems*, (Preprint), 1-13.

López, C., Codocedo, V., Astudillo, H., & Cysneiros, L. M. (2012). Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. *Science of Computer Programming*, 77(1), 66-80.

Martin, R. C. (2002). *Agile software development: principles, patterns, and practices*. Prentice Hall.

Motik, B., Glimm, B., Stoilos, G., Horrocks, I., & Shearer, R. (2011). *HermiT*. Retrieved September 15, 2018, from https://protegewiki.stanford.edu/wiki/HermiT

Stanford Center for Biomedical Informatics Research. (2008). *Using DL reasoners in Protege-OWL*. Retrieved September 15, 2018, from https://protegewiki.stanford.edu/wiki/Using_Reasoners

Stanford Center for Biomedical Informatics Research. (2016). *Protégé'*. Retrieved September 15, 2018, from https://protege.stanford.edu/

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: principles and methods. *Data & Knowledge Engineering*, *25*(1-2), 161-197.

Tang, A., Babar, M. A., Gorton, I., & Han, J. (2006). A survey of architecture design rationale. *Journal of systems and Software*, 79(12), 1792-1804.

Tang, A., Liang, P., & Van Vliet, H. (2011, June). Software architecture documentation: The road ahead. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on* (pp. 252-255). IEEE.

University of Oxford (n.d.). *HermiT OWL Reasoner*. Retrieved September 15, 2018, from http://www.hermit-reasoner.com/

W3C (n.d.a). *Semantic Web*. Retrieved September 15, 2018, from https://www.w3.org/standards/semanticweb/

W3C (n.d.b). *W3C Semantic Web*. Retrieved September 15, 2018, from http://www.w3.org/2001/sw

Zhao, Y., Dong, J., & Peng, T. (2009). Ontology classification for semantic-web-based software engineering. *IEEE Transactions on Services Computing*, 2(4), 303-317.

Bibliographic information of this paper for citing:

Bhatia, M.P.S., Kumar, Akshi, & Beniwal, Rohit (2018). "Ontology driven software development for automated documentation." *Webology*, 15(2), Article 174. Available at: http://www.webology.org/2018/v15n2/a174.pdf