sciendo RIGA TECHNICAL UNIVERSITY

Applied Computer Systems

ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
May 2019, vol. 24, no. 1, pp. 38–48
https://doi.org/10.2478/acss-2019-0006
https://content.sciendo.com

A Semantic Retrieval System for Case Law

Esingbemi Princewill Ebietomere[1], Godspower Osaretin Ekuobase[2*]
[1, 2] *Department of Computer Science, University of Benin, Benin City, Nigeria*

***Abstract* – Legal reasoning, the core of legal practice in many countries, is "stare decisis" and its soundness is usually strengthened by relevant case law consulted. However, the task of relevant case law access and retrieval is tiring to legal practitioners and constitutes a serious drain on their productivity. Existing efforts at addressing this problem are conceptional, restrictive or unreliable. Specifically, existing semantic retrieval (SR) systems for case law are desirous of exceptional retrieval precision. Ontology promises to meet this desire, if introduced to the SR system. As a consequence, an ontology-based SR system for case law has been built using the systems analysis and design methodology. In particular, the component-based software engineering and the agile methodologies are employed to implement the system. Finally, the search and retrieval performance of the resultant SR system has been evaluated using the heuristics evaluation method. The retrieval system has shown to have a search and retrieval performance of about 94 % precision, 80 % recall and 84 % F-measure. Overall, the paper implements the SR system for case law with excellent precision and affirms the superiority of ontology approach over other semantic approaches to SR systems for document retrieval in the legal domain.**

***Keywords* – Case law, document retrieval, heuristic evaluation, semantic retrieval.**

I. Introduction

Law has remained an indispensable part of man and every society, and organisation has not only become a product of law but is now also intrinsically and intricately operated by it. The intricate nature of law [1]–[5] has made the administration of justice, interpretation of law, trial of offenders and adjudication complex and time consuming [6], [7]. Computational law attempts to leverage the complexity and inefficiency bedevilling the legal system using computing [3], [8]; with a view to complementing or replacing legal experts [8]–[15].

The success achieved in the field of Artificial Intelligence (AI) particularly in building expert systems in various domains such as medicine and biotechnology amongst others made it apparent that human expertise could be substituted with systems codifying the human expert's know-how. The legal profession did not also escape this admiration for experts' know-how codification [9] and, as a result, the research efforts in computational law were geared towards codifying legal expert's know-how [1], [11], [13], [16]–[18]. Codifying legal expert's know-how is salutary but the adoption of its resultant systems will be seriously threatened by the nature of law and the legal practice [5], [19]. Basically, rules and discretions are two "desiderata" deployed in legal practice [6], [7], [20], [21]. Codifying rules may be easy but that of discretion is herculean [5], [22]. Exercising discretion particularly in the legal profession is not an easy task because it is strongly dependent on plain wisdom, experience, contextual and socio-political influences [2]–[6]; and these factors are diverse and vary widely across people, culture and time in history. This may bring the systems that codify legal expert's know-how to infamy.

Furthermore, these systems may receive resistance from legal experts because of the fear of being replaced by them in addition to the fear of leaving the legal practice in the custody of non-legal experts. We may, thus, be faced with a scenario where systems of codified legal expert's know-how are developed with a huge amount of money and effort but never used; for no matter how beautiful a technology is, if rejected, it is as good as not being in existence [23]. It behoves computational law experts, therefore, to pursue building systems that will complement rather than replace the legal experts in order to avoid building systems that may never be used.

This paper though frowns at building systems to replace legal experts semantically represents an important instance of legal information – case law; since semantic representation of legal information is critical to building systems that can complement or replace legal experts [24]. This is not the first time case law is semantically represented [1], [11], [16], and [25]. However, some of these prior semantic representations were geared towards codifying the case law construction process [1], [14], and [17]. However, case law as a legal construct is not only a function of legal rules but also a function of the skill set, experience and plain wisdom of the legal experts [6], which make the codification of its construction process demanding and unreliable. Consequently, this paper does not toe the path of the likes of [1], [14] and [17].

Legal reasoning, a critical part of legal practice, is strongly case-based, i.e., "stare decisis" [3], [6], [14], [18], and, thus, legal reasoning and judicial verdicts are both strengthened but further complicated by available case law that obviously increases with time in every judicial system. The study [26] established that the efficiency and the effectiveness of legal reasoning processes and judicial verdicts were influenced by how case law was stored, accessed and retrieved; and clamoured alongside the likes of [2], [3], [8], [13], [14], [18], and [24] for a semantic representation of legal information. These experts showed that only such a representation would allow for an excellently efficient and effective processing or handling of legal information by both man and machine.

A typical case law consists of two major parts: the "ratio decidendi" and the "obiter dicta" [1], [6], [7]. The ratio

* Corresponding author's e-mail: godspower.ekuobase@uniben.edu

decidendi encompasses both the principles of law on which a judicial verdict was based and the verdict itself; while the obiter dicta hold the facts that were not considered in a case to arrive at a judicial decision. Some of the existing works that involve semantic representations of case law [1], [11], [12], [16], [25] only considered case law in either of its parts – ratio decidendi or obiter dicta. This paper semantically represents case law without excluding any of its parts.

The representation mode of case law (or any document repository) defines the means of retrieving them, i.e., the retrieval system. In many countries, case law retrieval systems are predominantly manual, or where electronic, such a system is syntactic in nature. Manual retrieval system is cumbersome and tiring to legal practitioners. The problem with syntactic retrieval system is that most of the search results are irrelevant and so much man hour is wasted on manually screening the search result to obtain the relevant documents [26], [27]. Obviously, these retrieval systems constitute a serious drain on the productivity of legal practitioners. A semantic retrieval system for case law is therefore imperative for efficient and more effective legal practise.

The issue of effective and efficient storage, access and retrieval of information is particularly of great concern to the domain of law not only because this domain is purely information driven but also because the domain guarantees the sanity and sanctity, and hence the survivability of the society or organisation it operates. Besides, in line with the popular slogan that "Justice delayed is justice denied", and for the fact that the soundness of legal reasoning and judicial verdict in many countries is a function of the quality of information at the disposal of legal experts, the efficiency and effectiveness of legal information retrieval are grave. Moreover, the fact that legal reasoning and judicial verdict can be based on existing case law – a critical legal information – which increases with time, makes an efficient and effective case law retrieval system of colossal value to legal experts.

As a consequence, this study has designed, implemented and evaluated a semantic retrieval system for case law. The semantic retrieval system is a desktop application and hence does not concern itself with the security, reliability and scalability issues associated with large-scale distributed systems. This paper also restricts itself to textual information since law is basically text-based.

The remaining part of the paper is organised as follows. The next section holds the review of related semantic information retrieval systems in the domain of law. While Section 3 holds the proposed system's architecture, specification and design, Section 4 describes the materials and methods of implementation. Section 5 provides the evaluation of the resultant semantic retrieval system. Section 6 holds the conclusion and suggestion for further studies.

II. Related Work

The study [1] designed a conceptual retrieval system for case law with the objective of semantically retrieving arguments (ratio decidendi) and parts of argument from case law. The author employed Toulmin pattern of argument (a Toulmin model for argument representation) and was able to demonstrate the feasibility of conceptual retrieval of argument from case law. The system was however not implemented and thus its search and retrieval performance could not be ascertained. The information retrieval (IR) system, though semantic, only considered the ratio part of case law, which limited the information need of case law searchers.

The authors of the research [28] developed a system for managing case law. The objective of their work was to provide the user with the previous case law similar to a particular case law – the search item. They employed Case Retrieval Nets (a case-based reasoning technology) to index cases and used a combination of information extraction techniques with ontology to conceptualise the domain for knowledge sharing. This work, however, restricted the search need of case law searchers to history of known case law. The retrieval performance of the system was not evaluated.

The authors of the study [11] developed a History Assistance System. Their objective was to extract information such as named entities and judges' ruling in a case law from a citator database and to determine which cases in the same appellate chain were immediately impacted by the rulings. They employed a natural language approach (grammar and lexicon) and a machine learning technique (Support Vector Machine). This work also restricted the search need of case law searchers to history of known case law.

The studies [12] and [25] (both works appear technically equivalent) developed a system for the retrieval of tort/liability case law for the Netherlands using thesaurus. Thesaurus is a weak conceptualisation of domain concept and this obviously accounts for the poor correctness or reliability of the retrieval system.

The authors of the research [27] proposed a legal ontology framework. The objective of the work was to make available a robust ontology for legal information (normative documents and judicial cases) retrieval. They posited that ontology framework could enhance the retrieval precision of legal information though did not put this into practice or use.

Overall, the existing Semantic Information Retrieval Systems for case law are, on the one hand, conceptional and, on the other hand, unreliable or restrict user search needs. This paper uniquely designed and implemented a reliable Semantic Information Retrieval System for case law with no restriction on user search needs. This system will hereafter be referred to as "Law-Torch".

III. The System Architecture and Design

The necessity of prompt and accurate retrieval of case law by legal practitioners coupled with the limitations of existing case law retrieval architectures – restricted user need, imprecise and incomplete retrieval of case law birthed the Semantic Information Retrieval (SIR) architecture named "Torch". The proposed Torch architecture depicted in Fig. 1 is generic semantic retrieval architecture.

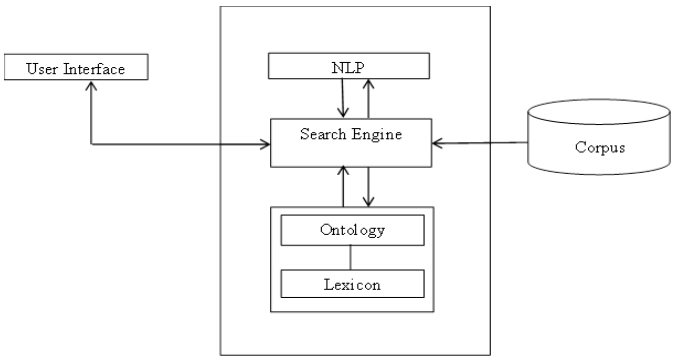

Fig. 1. The proposed Torch architecture.

The proposed Torch architecture consists of the five main parts – User Interface, Natural Language Processor (NLP), Search Engine, Knowledge Base (Ontology and Lexicon) and Corpus – highlighted as follows:

i) **User Interface**: the user interface allows the user to enter his or her information need in natural language and allow the user to view the search result. The information need supplied by the user is sent to the search engine as a character string.

ii) **Search Engine**: the Torch search engine is the hub of the architecture. The search engine sends the user information need to the NLP for processing and query generation, collects the query and matches them with the knowledge base and thereafter retrieves from the corpus the case law returned by the matching process. This retrieved case law is then returned as the search result for display on the user interface.

iii) **NLP**: NLP handles the transformation of the user information need to query in Torch. The output of this process is returned to the search engine as query – a set of concepts.

iv) **Knowledge Base**: the knowledge base here is composed of ontology and lexicon. The role of ontology in Torch is that of an index to the Corpus while the lexicon is used for disambiguation (handles polysemy and synonymy) of concepts in the ontology.

v) **Corpus**: this is simply the electronic repository of case law usually in a particular format.

In particular, the functionality of the search engine – the architecture's hub – was specified using pseudocode as captured in Algorithm 1.

A. The Law-Torch System's Specification

Algorithm 1 specifies the implementation of the proposed Torch architecture. The pseudocode was semantically spiced using comments. For example, the 13th line of the pseudocode is used to track the number of concepts in query returned by the NLP.

```
1.  need: String
2.  sentence, result, query, case: <String>;
3.  min, minInt, i, j, querySize: int;
4.  countMatches: <int>;
5.  caseName: <String<String>>;
6.  need=getUserNeed();
7.  invoke(NLP);          //A natural language processor component
8.  Do using NLP {
9.        sentence = annotate(need);
10.       sentence = tokenize(sentence);
11.       query = lemmatize(sentence);
12. }
13. querySize = query.size();      // get number of concepts in query
14. loop i = 1 to querySize::      //get number of matches for each concept
15.   match(query[i] with ontologyDir) →countMatches[i];
16.   loop i = 1 to querySize:: //search for concept in ontology and retrieve matches
17.     loop j = 1 to countMatches[i] ::
18.       match(query[i] with ontologyDir)
      →caseName[i].addMatches();
19. min = countMatches[1];  minInt = 1;
20. loop i = 1 to querySize::      // get concept with the least set of matches
21.       if (countMatches[i]< min){
22.          min = countMatches[i];
23.          minInt =i;
24.       }
25. case = caseName[minInt];       //least set of case names
26. j=0;
27.   while (++j<=countMatches[minInt])::
28.       foreach case[j]: caseName[i]:: //case[j] is a case name instance in
      case
29.          loop i = 1 to querySize::
30.              result.addCase();    //result= ∩_{i=1}^{querySize} caseName[i];
31.  display(result);
```

Algorithm 1: A pseudocode for implementing the proposed Torch architecture.

Details of the implementation of the proposed Torch architecture for semantic retrieval of case law was exposed using the Unified Modelling Language (UML) on the four basic design views of software systems, namely: (i) functional view, e.g., use case diagram, (ii) static structural view, e.g., class diagram, (iii) behavioural (dynamic structural) view, e.g., sequence and activity diagrams, (iv) architectural view, e.g., component and deployment diagrams, which are captured in the following sub-sections.

B. The Law-Torch System's Use Case Diagram

The use case diagram shown in Fig. 2 is used to describe the functionality of the proposed Torch architecture for implementation from the user's perspective.

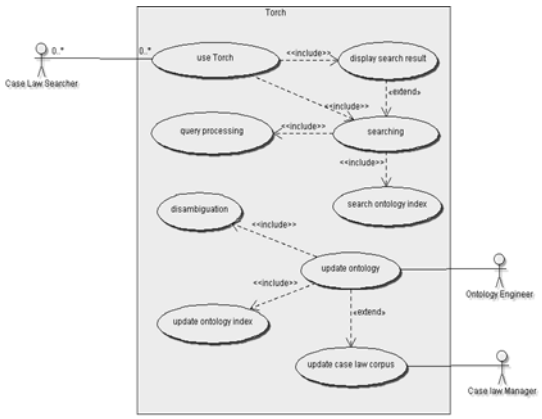

Fig. 2. Use case diagram for the Law-Torch system.

The use case diagram consists of three actors represented by stick persons: “Case Law Searcher”, “Ontology Engineer” and “Case Law Manager”. The use case diagram also consists of several operations represented by use cases (use Torch, display search result, query processing, searching, search ontology index, disambiguation, update ontology, update ontology index and update case law corpus).

The Case Law Searcher triggers the “use Torch” operation, which consists of “display search result” and “searching” operations. The “display search result” operation extends “searching” operation indicating that the search result cannot be displayed until “searching” operation is completed. The “searching” operation is made up of the “query processing” and the “search ontology index” operations and returns the result from these operations to the “display search result” operation. The “searching” operation implicitly gets the user need as a string of character, transfers it to the “query processing” operation, which in turn returns a set of concepts, which is then used to search the ontology index.

As a maintenance user, the Ontology Engineer updates the ontology as more case law is generated. The “update ontology” operation includes “disambiguation” and “update ontology index” operations. Since the ontology is case law corpus dependent, the “update ontology” operation depends on an updated case law corpus; hence, “update ontology” operation extends the “update case law” operation. The case law corpus update is performed by the case law corpus manager.

C. The Law-Torch System’s Class Diagram

Figure 3 depicts the class diagram of the Law-Torch system – a static view of the system in terms of its constituent classes and their relationships (e.g., association and generalisation). Five major classes as shown in Fig. 3 constitute the system. The “GUI class” is responsible for the entering of information need and displaying of the search result. This class communicates with the “SearchEngine” class in two ways – sends the user information need to the SearchEngine and receives the result from the SearchEngine for display. The “SearchEngine” class aggregates the “Searcher”, “LuceneConstants”, and the “TestFileFilter” class to carry-out its function. The role of the “Searcher” is to match query with an ontology index and return documents in order of relevance. The “LuceneConstant” class defines the retrieved document file name and path name. The TestFileFilter ensures that the retrieved documents are of type text (.txt).

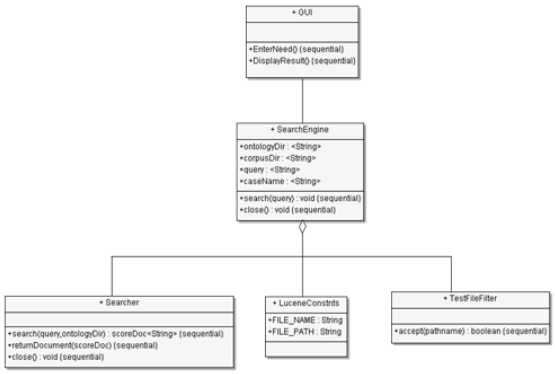

Fig. 3. Class diagram for the Law-Torch system.

D. The Law-Torch System’s Sequence Diagram

As exposed in Fig. 2, two basic external operations – the search and ontology update operations – can be performed with the system at execution. Thus, two sequence diagrams are used to describe the Torch system as shown in Figs. 4 and 5. While Fig. 4 describes the behaviour of the system when used by the Case Law Searcher for search, Fig. 5 describes its behaviour when used by the Ontology Engineer for ontology update.

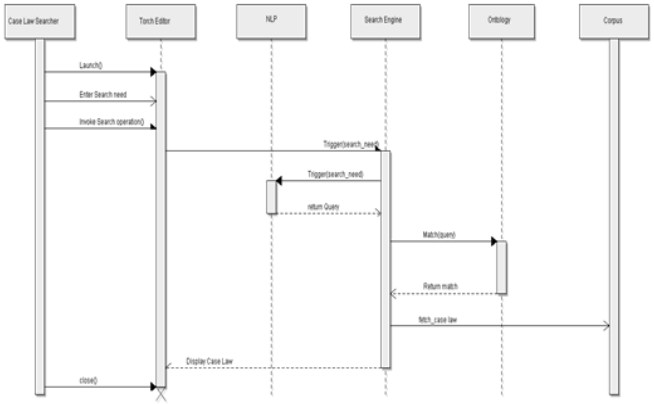

Fig. 4. Sequence diagram for the Law-Torch system’s search use.

As shown in Fig. 4, the Case Law Searcher launches the “Torch Editor” and enters his/her information need in natural language. When he/she hits the search button on the editor, the search need is sent by the “Search Engine” to the “NLP” object to process the information need. Once this process is completed by the NLP, the processed information need (query) is sent back to the Search Engine which activates the “Ontology” to check for matches between the query elements and the ontology index. The matches returned are then used by the Search Engine to fetch the relevant case law from corpus, which is returned to the Torch Editor for display.

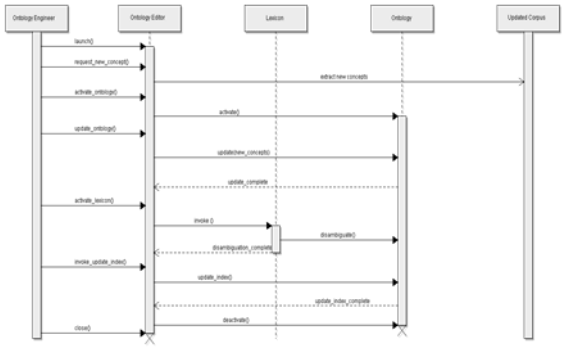

Fig. 5. Sequence diagram for the Law-Torch system's ontology update use.

As shown in Fig. 5, the "Ontology Engineer" launches the "Ontology Editor" and requests new concepts to perform update. These concepts are extracted from the "Updated Corpus". These concepts are then added to the "Ontology". Once the ontology update is complete, a "Lexicon" is invoked to disambiguate the newly added concepts. On completion of the disambiguation process, the ontology index is updated and thereafter the ontology editor is closed by the "Ontology Engineer".

E. The Law-Torch System's Component Diagram

The component diagram is used to show the various components that make up a system, their interactions, and dependency. Figure 6 shows various components of the Law-Torch system.

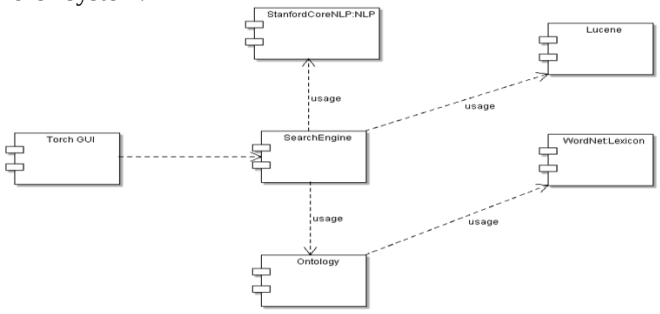

Fig. 6. Component diagram for the Law-Torch system.

The "Torch GUI" component is the interface via which a user enters information need and views the displayed results. The "Search Engine" component uses the Stanford CoreNLP, Lucene and Ontology components. The Stanford CoreNLP component is used to transform the user information need to query. The Stanford CoreNLP is chosen among other open source NLP tools, such as Open NLP and NLTK, because it is a light-weight Java-based simple annotation pipeline, platform independent and integrates seamlessly with Eclipse (our IDE of choice). Lucene is used by the SearchEngine to match the user query to the ontology index for the retrieval of relevant case law. Lucene engine is opted for among other engines like Terrier and Nutch because it is platform independent, popular with strong community support and more importantly it supports ontology indexing. The "Ontology" component provides the index to be searched for retrieval of case law. The "Ontology" component uses another component called WordNet (lexicon) for concept disambiguation. WordNet is chosen among other lexicons like dictionaries because it can handle disambiguation better than dictionaries for the fact that it has its words arranged as synset (synonyms) – the reason why WordNet is sometimes referred to as a lexical ontology.

IV. Materials and Method of System Implementation

After the specification and design of the Torch architecture, the next step is to translate these artefacts into a working system. The Torch architecture is component based, hence the Component Based Software Engineering (CBSE) – a sub-discipline of Software Engineering that provides methods, models and guidelines for the developers of component-based systems [29], [30] – has been used in the implementation of the Torch architecture. CBSE emphasises the Component-based Development (CBD) that deals with developing systems by making use of pre-existing components [31], [32]. The primary goal of the CBD is to create a complex architecture by reusing a smaller and more manageable software element [33]. This approach in many cases has proven to simplify software design; it reduces time-to-market, lowers cost of development, allows for effective management of complexity, increases productivity, improves quality, brings about a greater degree of consistency, eases maintenance, and widens the range of usability [29], [33].

Using a CBD approach, it is important to note that much implementation effort in system development is no longer necessary, but effort is shifted to dealing with components, locating the components, selecting the components that are most appropriate for a specific task, integrating and testing the components [31]. The CBD approach can be used with either of the two main categories of software development process model, i.e., the sequential model of which a classical example is the waterfall model; and the evolutionary (iterative and incremental) model of which an example is the agile model [31], [34]. Though no software development process is a failure and none is a silver bullet – each has its strengths and weaknesses as well as project domain or nature of software project it can best handle [29], [35]. The agile model has been selected because it is best suited for visible systems, low-risk projects, projects with blurred and unstable requirements, small-to-medium sized projects and projects that are time-to-market driven as shown in Table I [35] – these attributes are inherent in the Law-Torch system's project.

TABLE I
Software Projects and the Basic Software Development Approaches [35]

Agile Software Development Approach	**Traditional Software Development Approach**
Visible systems	Legacy/Embedded systems
Low risk projects	High risk projects
Blurred and unstable requirements	Explicit and fairly stable requirements
Small-to-medium sized projects	Large and complex projects
Time-to-market driven	Product quality driven

The tailored agile approach, in Fig. 7 adapted from [35], is the software development process for the Law-Torch system, which explained as follows:

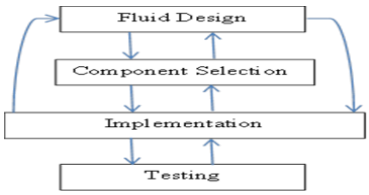

Fig. 7. An agile implementation model.

The design phase in Fig. 7 is labelled "Fluid Design" because it is non-static and as such can be easily adjusted if a need arises. Fluid design is followed by component (software component) selection. Though there are different definitions adduced to the concept "component", this work aligns itself with the widely accepted definition of the concept, which sees a software component as a unit of composition with contractually specified interfaces (has defined API and all assumptions in which the component can work) and explicit context dependencies [29], [30]. After selecting the necessary components, implementation is carried out by gluing all the components selected into a single working unit. Should there be issues as regards getting the selected components to work together, the components are reviewed and this could lead to adjusting the design to portray the new stance. After a successful integration (implementation), it is necessary to test the system to ensure all the components glued together are working as expected. Issues arising at the testing phase could lead to reviewing the implementation, which in turn could lead to reviewing the selected components that could result in adjusting the design. The process is performed iteratively until a satisfactorily working system is implemented. The afore-described agile process has been judiciously followed in building the Law-Torch system.

In the development of the Law-Torch system, hardware and software tools have been employed. These tools and the roles they played in the system's implementation are discussed as follows.

- **Hardware tools**: the hardware is basically a personal computer (PC), which serves as the host to all the software tools used in the research. The PC is a Lenovo notebook with the following specification:
  Processor: Intel(R) Pentium(R) CPU N3540 @ 2.16 GHz 2.16 GHz
  Installed Memory (RAM): 4.00 GB
  System Type: 64-bit Operating System, x64-based Processor
  Hard Disk space: 1 TB
- **Software Tools**: the software tools used for building the Law-Torch system are discussed under the Operating Systems, Development Platform, Language, Integrated Development Environment (IDE), Editors and Component as follows:
  i) **Operating System**: Microsoft Windows 8 Pro edition has been used. The Operating System enables the Torch application and other software tools to interact with the computer hardware devices.
  ii) **Development Platform**: Java and .NET are two outstanding development platforms for enterprise application ([35], [36]). Java EE has been opted for because of prior Java programming skill.
  iii) **Language**: language here includes programming language and modelling language. Java has been used for knitting all the components of the Law-Torch system together. Java has been opted for by virtue of the fact that the development platform of choice for the Law-Torch system is Java EE.
  iv) **Integrated Development Environment (IDE)**: IDE makes programming easy, lithe and interesting [35], [36]. There are several IDEs that support Java; these include Eclipse, Netbeans, IntelliJ IDEA. The Eclipse IDE has been chosen among other free IDEs like NetBean, and IntelliJ IDEA because of its cross-platform capability, extensible tool support, extensive help and documentation, support for desktop Java application and, most importantly, seamless interaction among the selected components for the implementation of the Law-Torch system.
  v) **Components**: The software components used by the Law-Torch system include: Natural Language Processor, Lexicon and Search Engine. The natural language processor has been used to transform the case law searcher need to query, and it is popular among the Natural Language Processing tools such as OpenNLP, NLTK, UIMA, Stanford CoreNLP and GATE [37]. Stanford CoreNLP has been settled for because it is Java-based, simple to use and light-weight [37]. Moreover, the Stanford CoreNLP is open source with rich documentation and viable user support community. Lexicons are linguistic resources that contain words and their meanings. Lexicon is used in the Law-Torch system for ontology disambiguation and enrichment. Examples of lexicons are dictionaries and WordNet. WordNet has been chosen among other lexicons like dictionaries because it can handle disambiguation better than dictionaries for its words are arranged as synset (synonyms) – the reason why WordNet is sometimes referred to as a lexical ontology.
  vi) **Search Engine**: There is a myriad of search engines [38], [39]. The role of the search engine in the Law-Torch system is to enable the search capability of the system. The choice of search engine component for the Law-Torch system has been incident on high performance, light-weight, platform independent, ease of use, strong community support and ability to handle ontology indices. Apache Lucene – a powerful Java-based search engine framework – has been chosen among its rivals such as Nutch and Solr ([38]) because it is best suited for adding search functionality to desktop applications and far more light-weight as compared to its rivals, which are full-fledge search

applications built on Lucene. Moreover, the Apache Lucene search engine works seamlessly with the platform of choice for the Law-Torch system. Apache Lucene has been employed to enable the search of query in the ontology index.

vii) **Ontology**: The ontology selected for the Law-Torch system is topic ontology for case law – TONCL [40]. This ontology has been selected because of its ontological commitment of case law retrieval.

A. Building the Law-Torch System

The components used in the Law-Torch system include pre-existing reusable components and a self-tailored component (ontology). The pre-existing reusable components are Apache Lucene 3.6.2, Stanford CoreNLP 3.6.0 and WordNet 2.0. To make the system work as a single unit, Eclipse IDE 4.5 has been downloaded and installed. The Eclipse IDE requires an appropriate Java Run-time Environment (JRE)/Java Development tool-Kit (JDK), which has also been downloaded and installed. JRE consists of libraries and files used by Java Virtual Machine (JVM) – an abstract machine; at run-time. JDK consists of JRE, compiler and tools (JavaDoc and Java Debugger) to create and compile programs written in Java. JRE/JDK 8 has specifically been used.

To use Apache Lucene version 3.6.2 in Eclipse IDE, several necessary Java Archive (jar) files such as the lucene-core-3.6.2.jar and lucene-queryparser-3.6.2.jar have been imported for configuration in Eclipse IDE. A case of the configuration build path of Apache Lucene 3.6.2 is shown in Fig. 8.

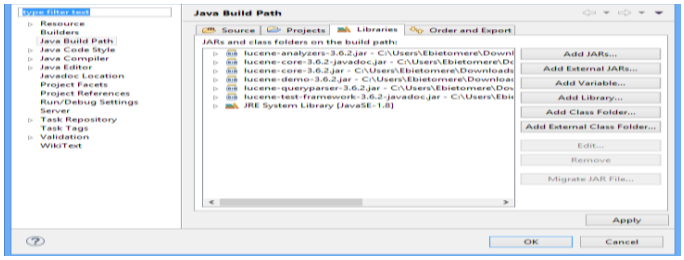

Fig. 8. Lucene configuration build path in Eclipse.

To use the Stanford CoreNLP version 3.6.0 component in the Eclipse IDE, files like stanford-corenlp-3.6.0.jar and stanford-corenlp-3.6.0-models.jar have been required for configuration. A case of the configuration build path of Stanford CoreNLP 3.6.0 via the Eclipse IDE is shown in Fig. 9.

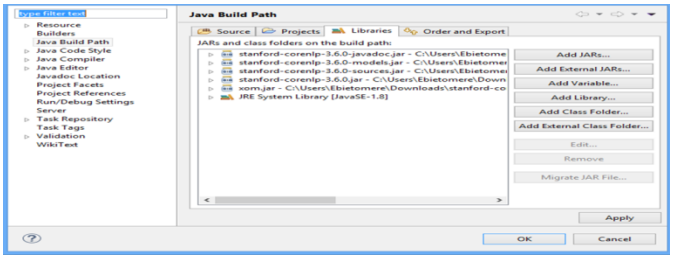

Fig. 9. Stanford Corenlp configuration build path in Eclipse.

The ontology component, a very important component of the Law-Torch system employed, has been the TONCL [40]. Though there are other several ontologies in existence for the legal domain [24], [26], [41] and [42], no one could be reused because none was in alignment with our ontological commitment of case law retrieval.

After successful configurations, the components for the Law-Torch system have been knitted together through their Application Programming Interface (API) as specified in Algorithm 1, using the Java programming language. The knitting has implemented the Law-Torch system; and has been successful as shown in Fig. 10. Figure 10 depicts the successful integration of the components of the Law-Torch system.

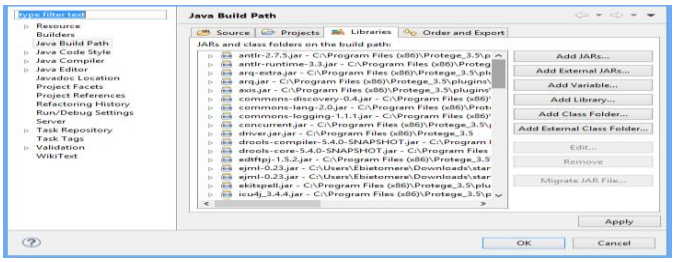

Fig. 10. Law-Torch component integration build path.

Several challenges have been faced in the course of making all the components of the system work as a unit. The questions as to what component tool to use to perform a particular task, how well the component tool can perform the task (maturity of the tool, documentation of the tool and popularity of the tool etc.), how to configure the component tool to work with other component tools and the compatibility (in terms of version, operating system, and language etc.) of the component tool with others have been a menace. There have been cases where some component tools have been jettisoned after considerable effort and success because they have not been performing well enough, the desired format of input/output has been lacking in the tool, poor documentation for conflict resolution, heavy weight (tools with multiple scale-out may have unused part interfering with the running of the system) thereby weighing the system down, outdated, i.e., not currently maintained, files for configuration are not readily available, or compatibility issues. Painfully, some of these component tools have to be mastered before you can ascertain if they are used or discarded.

V. System Evaluation and Result

It has been pertinent to evaluate the Law-Torch system to ascertain its retrieval performance. To do this, the Heuristic Evaluation Method has been employed because it is reliable though expensive [43], [44]. Three evaluators – legal practitioners with less experienced practitioners called to the Nigerian Bar in 2011 evaluated the system for relevance of retrieved case law for the same set of structured search needs covering topics in the area of land dispute, ownership, tenancy, and damage amongst others as shown in Table II. The search needs or queries have been jointly formulated by the evaluators to sufficiently capture varying search needs of legal experts in Nigeria. We then have made available hard copies of 280 Nigeria Supreme Court case law randomly selected from the pool of corpus used to create TONCL for the system's evaluation experiment, to each of the evaluators to manually and separately study and establish appropriateness of the given

case law to each of the search needs. Thereafter, the evaluators were exposed to the Law-Torch in a training session. Each evaluator then did their individual evaluation using the Law-Torch system and recorded their results between 24 February and 3 March 2017 and thereafter met as a team on 4 March 2017 to reconcile verdicts in consensus. The consensus verdict is reported in Table II.

TABLE II
CONSENSUS VERDICT FROM THE HEURISTIC EVALUATION

Search need	Case retrieved	Relevant case retrieved	No. of relevant case in the corpus
Cases on land dispute	Ogunmola v Eiyekole Olagbemiro v Ajagungbade Nwosu v Chukwumanjo Umeojiako v Ezenamuo Idundun v Okumagba Adomba v Odiese Are v Ipaye Adelaja v Fanoiki	Ogunmola v Eiyekole Olagbemiro v Ajagungbade Nwosu v Chukwumanjo Umeojiako v Ezenamuo Idundun v Okumagba Adomba v Odiese Are v Ipaye Adelaja v Fanoiki	11
Cases on damage paid on libel	Williams v DailyTimesNigeria	Williams v DailyTimesNigeria	1
Cases on general damages	AGO v Fairlakes Amaye v ARE Dumez v Ogboli AGB v Aideyan	AGO v Fairlakes Amaye v ARE Dumez v Ogboli AGB v Aideyan	4
Cases on land tenancy	Ogunmola v Eiyekole Olagbemiro v Ajagungbade Are v Ipaye	Ogunmola v Eiyekole Olagbemiro v Ajagungbade Are v Ipaye	4
Cases heard under justice Adolphus Godwin Karibi-Whyte	Animashaun v Olojo Aruna v State	Animashaun v Olojo Aruna v State	6
Cases on customary tenancy	Ogunmola v Eiyekole Olagbemiro v Ajagungbade Are v Ipaye	Ogunmola v Eiyekole Olagbemiro v Ajagungbade Are v Ipaye	3
Cases on land and ownership	Olagbemiro v Ajagungbade Nwosu v Chukwumanjo Umeojiako v Ezenamuo Tijani v Secretary Idundun v Okumagba Adomba v Odiese Are v Ipaye	Olagbemiro v Ajagungbade Nwosu v Chukwumanjo Umeojiako v Ezenamuo Tijani v Secretary Idundun v Okumagba Are v Ipaye	8
Cases that border on the constitution of Nigeria	AGA v AGF Tukur v State AGF v AGA Ogunmola v Eiyekole	AGA v AGF Tukur v State AGF v AGA	5
Cases on stay of execution	Agbaje v Adeoti Okoya v Santilli Mohammed v Lasisi Shodehinde v Islam	Agbaje v Adeoti Okoya v Santilli Shodehinde v Islam	3
Criminal cases that are on murder	Akilu v Fawehinmi Nkanu v State Adekunle v State Ukwunnenyi v State Ndu v State	Akilu v Fawehinmi Nkanu v State Adekunle v State Ukwunnenyi v State Ndu v State	6

Table II has four columns with the headings, "search need", "case retrieved", "relevant case retrieved" and "number of relevant cases in the corpus"; described as follows: for example, for the first user search need stated as "cases on land dispute", a total of eight cases have been retrieved, all the eight cases have been adjudged relevant by the evaluators, but there are a total of eleven cases in the corpus that border on this subject requested by the searcher according to the evaluators. Similar explanation follows for the other search needs.

The consensus verdict from Table II has been subsequently used to compute the system's search and retrieval performance using the system-based evaluation metrics [45]–[48] given in Equations (1)–(3).

$$Precision = \frac{Number\ of\ relevnt\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}; \quad (1)$$

$$Recall = \frac{Number\ of\ relevnt\ documents\ retrieved}{Total\ number\ of\ relevant\ documents\ in\ the\ corpus}; \quad (2)$$

$$F\text{-}measure = 2\frac{Precision \cdot Recall}{Precision + Recall}. \quad (3)$$

The results obtained after computation using the evaluation metrics defined by Equation (1)–(3) are shown in Table III.

TABLE III
COMPUTED RESULT

Search Need No.	Precision	Recall	F-measure
1.	1	0.73	0.843931
2.	1	1	1
3.	1	1	1
4.	1	0.75	0.857143
5.	1	0.3	0.461538
6.	1	1	1
7.	0.86	0.75	0.801242
8.	0.75	0.6	0.666667
9.	0.75	1	0.857143
10.	1	0.83	0.907104
Average	0.936	0.796	0.839477

For better appreciation, the Precision, Recall, Precision vs. Recall and Recall vs. Precision graphs have been plotted as shown in Figs. 11–14, respectively.

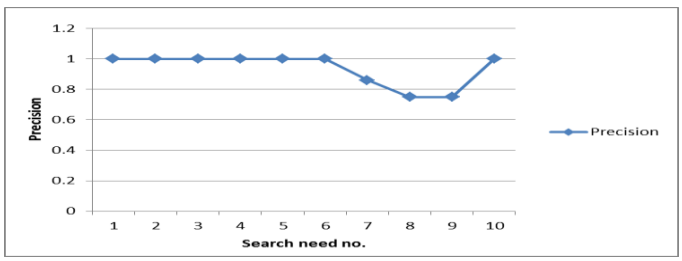

Fig. 11. Precision line graph of Law-Torch.

Figure 11 shows the line graph of precision over the ten search needs presented to the Law-Torch system by the case law searcher. The least precision values obtained are at points 8 and

9 with precision being around 0.75. While the maximum precision of one occurs most often, the worst case precision of about 75 % occurs less often. Figure 12 gives a clearer picture of the precision performance of the Law-Torch system by giving the line of best fit precision of the system.

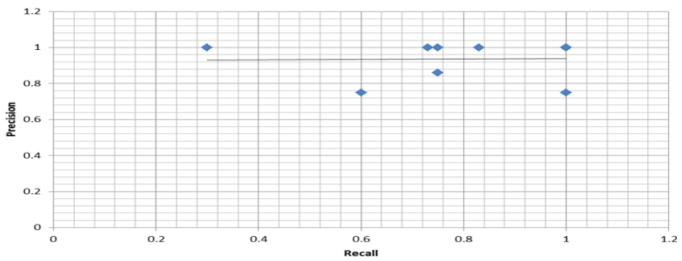

Fig. 12. Precision vs. recall scatter graph for Law-Torch.

Figure 12 is the scatter graph of linear precision for the Law-Torch. From the graph, it is evident that the average precision of the Law-Torch system is about 0.94 irrespective of the recall.

Figure 13 shows the recall line graph for the Law-Torch over the ten search needs presented to the Law-Torch system by the case law searcher.

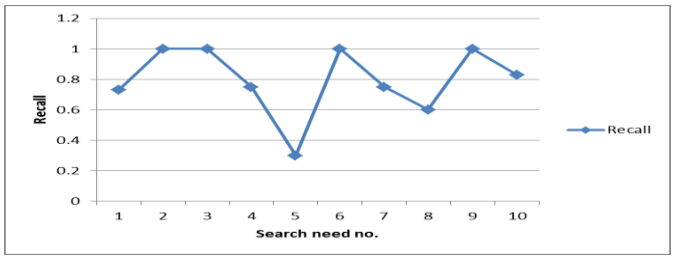

Fig. 13. Recall line graph of Law-Torch.

From Fig. 13, it is obvious that the recall at some points are as high as 1 and at a point as low as 0.3. This recall result is irregular and could be a possible source of improvement for the Law-Torch system. With this graph, the recall performance of the Law-Torch system could not be predicted; and we had to plot a scatter graph of precision vs. recall as shown in Fig. 14.

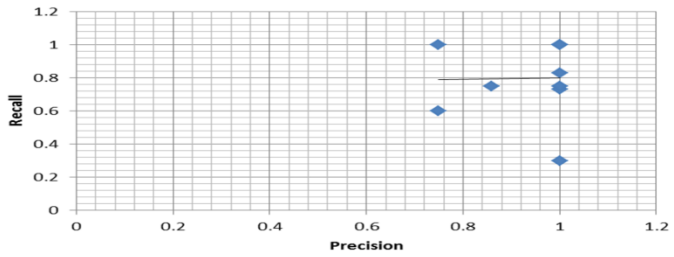

Fig. 14. Recall vs. precision scatter graph of Law-Torch.

Figure 14 is the scatter graph of recall on the vertical axis and precision on the horizontal axis. It is obvious from the graph that the recall performance of the Law-Torch system is about 0.8; an indication that the recall performance of Law-Torch system is about 80 %.

Thereafter, a linear graph of both precision and recall for the Law-Torch is plotted as shown in Fig. 15.

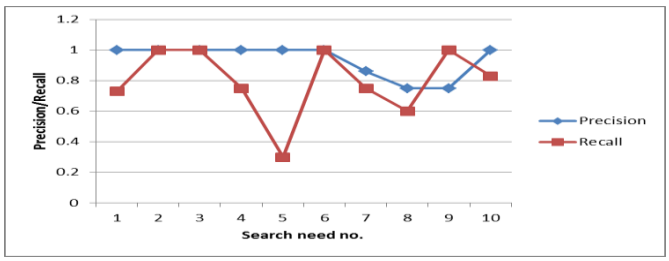

Fig. 15. Linear graph of precision and recall of Law-Torch.

It is evident from Fig. 15 that the precision of the Law-Torch system is consistently higher than its recall; a necessary attribute of semantic search systems.

The linear graph of precision, recall and F-measure has been plotted over the set of user need as shown in Fig. 16.

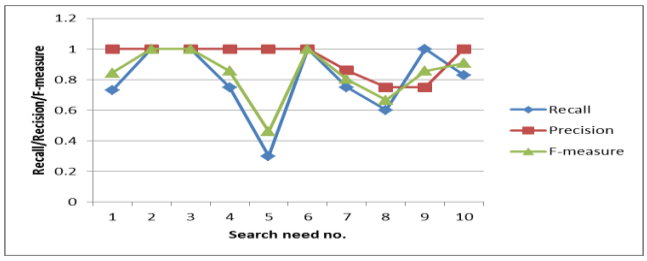

Fig. 16. Linear graph of precision, recall and F-measure for Law-Torch.

Figure 16 gives a complete description of the retrieval performance of the Law-Torch system with the introduction of F-measure, which leverages on the result of precision and recall to give the overall performance of the system. The F-measure is the harmonic mean of the precision and the recall of the Law-Torch system. The harmonic mean has been chosen in this case over other types of mean like the arithmetic mean and the geometric mean because it helps mitigate the effect of outlier. The bar graph of recall, precision and F-measure is also given in Fig. 17 to aid the comprehension of how the Law-Torch system performs relatively with the three basic retrieval metrics.

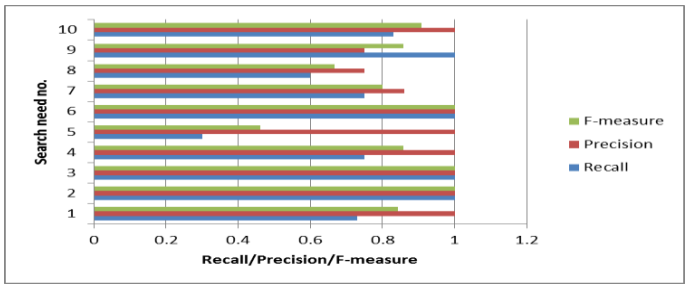

Fig. 17. A Bar graph of recall, precision and F-measure.

A. Result Interpretation

From Table III, it is easy to appreciate that the precision or retrieval reliability of Law-Torch is capable of 100 % precision, 70 % of the time with an average of about 94 %. For the recall, which is expectedly lower than precision, the Law-Torch retrieval system has a recall of 100 %, 40 % of the time with an average recall of about 80 %. This clearly out-performed the

semantic retrieval system in [11], which made use of natural language and machine learning approaches. In [11], a restricted case law retrieval system – history assistant system – had a maximum precision and recall of 94 % and 90 % respectively; which rarely occurred. Their average precision and recall were 89 % and 78 % respectively. Similar analysis holds for [12], [25], which used a thesaurus-based approach. We affirm, therefore, that an ontology-based approach is most appropriate for implementing a semantic retrieval system for case law.

It is also clear from Table III and Fig. 12 that the average precision of the Law-Torch system is about 0.94 – meaning for every search need posed to the system, 94 % of the retrieved case law is always relevant. The recall of about 0.80 means for every search need posed to the Law-Torch system, 80 % of the total relevant cases in the corpus are always retrieved. The F-measure of the system being about 0.84 means the average retrieval performance of the system is about 84 %. Again, it is evident that the system guarantees a search precision of 100 % most of the time (see Fig. 11). The implication of the above results is that the Law-Torch system is an excellent search and retrieval system for case law.

VI. Conclusion and Suggestion for Further Studies

Case law access and retrieval task constitutes a serious drain on the productivity of legal practitioners. However, existing efforts at relieving legal practitioner of this task are conceptional, restrictive or unreliable. We have shown that an ontology-based SR system is most appropriate to precisely meet the unrestricted search needs of case law searchers. Ontology-based semantic retrieval system for case law has been built. The system named Law-Torch has shown to have a search and retrieval performance of about 94 % precision, 80 % recall and 84 % F-measure – an indication that the Law-Torch is an excellent (case law) search system capable of boosting the productivity of legal practitioners. Furthermore, the research affirms the superiority of ontology over other semantic approaches for implementing document retrieval systems.

For real life deployment and use, the Law-Torch can be re-engineered into a secured and robust digital library of case law with its information base – case law corpus – expanded (and regularly updated) to include available case law. Productivity assessment of the use of the Law-Torch on legal practitioners should be carried out to gauge its impact on the productivity of legal practitioners.

References

[1] J. P. Dick, "Conceptual Retrieval and Case Law," ACM, pp. 106–115, 1991.

[2] J. Breuker, and R. Winkels, "Use and Reuse of Legal Ontologies in Knowledge Engineering and Information Management," ICAIL Workshop on Legal Ontologies and Web Based Legal Information Management. pp. 1–35, 2003.

[3] N. Love, and M. Genesereth, "Computational Law," *Proc. International Conference on Artificial Intelligence and Law (ICAIL '05)*, Bologna, Italy, pp. 205–209, 2005. https://doi.org/10.1145/1165485.1165517

[4] G. Lame, "Using NLP techniques to Identity Legal Ontology Components: Concepts and Relations," *Artificial Intelligence and Law*, vol. 12, no. 4, pp. 379–396, 2006. https://doi.org/10.1007/s10506-005-4160-3

[5] C. Stevens, V. Barot, and J. Carter, "The Next Generation of Legal Expert Systems: New Dawn or False Dawn?, " in M. Bramer, M. Petridis, and A. Hopgood (Eds.) *Research and Development in Intelligent Systems XXVII. SGAI 2010*, Springer-Verlag London, 2010. https://doi.org/10.1007/978-0-85729-130-1_33

[6] N. I. Aniekwu, *Legal Methodology and Research in Nigeria: An Introduction*. Mindex Publishing, 2001.

[7] O. Aigbovo, *Introduction to Nigerian Legal system*. Sylva Publishing Inc. 2009.

[8] H. Andersson, "Computational Law: Law That Works Like Software," In R. H. Lee (Ed), CodeX- The Stanford Center for Legal Informatics. codex.stanford.edu. p. 25, 2014.

[9] R. Gruner, "Thinking Like a Lawyer: Expert Systems for Legal Analysis," *Berkeley Technology Law Journal*, vol. 1, no. 2, pp. 259–328, 1986.

[10] D. H. Berman, and C. D. Hafner, "The Potential of artificial intelligence to help solve the crisis in our legal system," *Communications of the ACM*, vol. 32, no. 8, pp. 928–938, 1989. https://doi.org/10.1145/65971.65972

[11] P. Jackson, K. Al-Kofahi, A. Tyrrell, and A Vachher, "Information Extraction from Case Law and Retrieval of Prior Cases," *Artificial Intelligence Journal*, vol. 150, no. 1–2, pp. 239–290, 2003. https://doi.org/10.1016/S0004-3702(03)00106-1

[12] E. M. Uijttenbroek, A. R. Lodder, M. C. A. Klein, W. G. Rildeboer, W. V. Steenbergen, R. L. L. Sie, P. E. M. Huygen, and F. van Harmelen, "Retrieval of Case Law to Provide Layman with Information about Liability: Preliminary Results of the BEST-Project ," *Computable Models of the Law, LNCS*, vol. 4884, pp. 291–311, 2008. https://doi.org/10.1007/978-3-540-85569-9_19

[13] A. Wyner, and R. Hoekstra, "A Legal Case OWL Ontology with an Instantiation of Popov v. Hayashi," *The Knowledge Engineering Review*, vol. 14, no. 2, pp. 1–24, 2010.

[14] A. Wyner, R. M. Palau, M. F. Moens, and D. Milward, "Approaches to Text Mining Arguments from Legal Cases", *in Lecture Notes in Computer Science LNCS*, vol. 6036, pp. 60–79, 2010. https://doi.org/10.1007/978-3-642-12837-0_4

[15] V. M. Naik, and S. Lokhanday, "Building a Legal Expert System for Legal Reasoning in Specific Domain- A Survey," *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 4, no. 5, pp. 175–184, 2012. https://doi.org/10.5121/ijcsit.2012.4514

[16] E. L. Rissland, D. B. Skalak, and M. T. Friedman, "*Case Retrieval through Multiple Indexing and Heuristic Search," IJCAI*, vol. 2, no. 901–908, 2003.

[17] E. L. Rissland., K. Ashley, and K. Branting, "Case Based Reasoning and Law," *The Knowledge Engineering Review*, vol. 20, no. 3, pp. 293–298, 2006. https://doi.org/10.1017/S0269888906000701

[18] P. Condliffe, B. Abrahams, and J. Zeleznikow, "An OWL Ontology and Bayesian Network to Support Legal Reasoning in the Owners Cooperation Domain," 2010. Available from: http://www.abs.gov.au, [Accessed 14 August 2013]

[19] R. L. Marcus, "The Impact of Computers on the Legal Profession: Evolution or Revolution?" Northern University Law Review. vol. 102, no. 4, pp. 1827–1865, 2008.

[20] K. B. Mensah, "Legal Control of Discretionary Powers in Ghana: Lessons from English Administrative Law Theory", *Africa Focus*, vol. 14, no. 2, pp. 119–140, 1998. https://doi.org/10.21825/af.v14i2.5549

[21] W. Lacey, "Judicial Discretion and Human Rights: Expanding the Role of International Law in the Domestic Sphere," *Melbourne Journal of International Law*, vol. 5, no. 25, 2004.

[22] J. Franklin, "Discussion paper: how much of commonsense and legal reasoning is formalizable? A review of conceptual obstacles," *Law, Probability and Risk*, vol. 11, no. 2-3, pp. 225–245, 2012. https://doi.org/10.1093/lpr/mgs007

[23] E. P. Ebietomere, and G. O. Ekuobase, "Issues on Mobile Agent Technology Adoption," *African Journal of Computing and ICT*, vol. 7, no. 1, pp. 21–32, 2014.

[24] C. N. Caralt, "Modelling Legal Knowledge through Ontologies. OPKJ: the Ontology of Professional Judicial Knowledge," Ph.D. Thesis, Departament de Ciencia Polıtica i Dret Public, Universitat Autonoma De Bercelona, p. 527, 2008.

[25] M. C. A. Klein, W. V. Steenbergen, E. M. Uijttenbroek, A. R. Lodder, and F. van Harmelen, "Thesaurus-based Retrieval of Case Law," *In the Proceedings of Jurix2006*, pp. 61–70, 2006.

[26] G. O. Ekuobase, and E. P. Ebietomere, "Ontology for Nigerian Case Laws," *African Journal of Computing and ICT*, vol. 6, no. 2, pp. 177–194, 2013.

[27] N. Zhang, Y. Pu, and P. Wang, "An Ontology-based Approach for Chinese Legal Information Retrieval," *In the Proceedings of Science, CENet2015*, Shanghai, China, vol. 259, 2015. https://doi.org/10.22323/1.259.0076
[28] M. Costa, O. Sousa, and J. Neves, "Managing Legal Precedents with Case Retrieval Nets," *The Twelfth Conference on Legal Knowledge Based Systems, JURIX 1999*, Nijmegen, pp. 13–22, 1999.
[29] I. Crnkovic, "Component-based Software Engineering – New Challenges in Software Development," *Software Focus*, vol. 2, no. 4, pp. 127–133, 2001. https://doi.org/10.1002/swf.45
[30] I. Crnkovic, , and M. Larsson, " Component-based Software Engineering – New Paradigm of Software Development," MIPRO, 13pp, 2001..
[31] I. Crnkovic, S. Larsson, and N. Chaudron, "Component-based Development Process and Component Lifecycle," *Journal of Computing and Information Technology*, vol. 13, no. 4, pp. 321–327, 2005. https://doi.org/10.2498/cit.2005.04.10
[32] I. Sommerville, *Software Engineering Eight Edition*. Pearson Education, p. 868, 2007.
[33] M. Novak, and I. Svogor, "Current Usage of Component Based Principles for Developing Web Applications with Frameworks: A Literature Review," *Interdisciplinary Description of Complex Systems*, vol. 14, no. 2, pp. 253–276, 2016. https://doi.org/10.7906/indecs.14.2.14
[34] V. Szalvay, "An Introduction to Agile Software Development," *Danube Technologies*, p. 11, 2004.
[35] G. O. Ekuobase, and E. A. Onibere, "Scalability of Web Services Solutions Built on ROA," *Canadian Journal of Pure and Applied Science*. SENRA: British Columbia, vol. 7, no. 1, pp. 2251–2270, 2013.
[36] G. O. Ekuobase, and I. Anyaorah, "Tail Tolerance of Web Services Solution Built on Replication Oriented Architecture (ROA)," *Canadian Journal of Pure and Applied Science*. SENRA: British Columbia, vol. 8, no. 2, pp. 2943–2954, 2014.
[37] C. D. Manning, M. Surdeanu, , J. Bauer, J. Finkel, J. B. Steven and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014. https://doi.org/10.3115/v1/P14-5010
[38] M. Khabsa, S. Carman, S. R Choudhury, and C. L. Giles, "A Framework for Bridging the Gap between Open Source Search Tools," In SIGIR 2012 Workshop on Open Source Information Retrieval, pp. 32–39, 2012.
[39] A. Trotman, C. L. A. Clarke, I. Ounis, S. Culpepper, M. Cartright, and S. Geva, "Open Source Information Retrieval: A Report on the SIGIR 2012 Workshop," *ACM*, vol. 46, no. 2, pp. 95–101, 2012. https://doi.org/10.1145/2422256.2422269
[40] E. P. Ebietomere, "A Semantic Retrieval System for Nigerian Case Law," Ph.D. Thesis, University of Benin, Benin City, p. 260, 2018.
[41] J. Breuker, R. Hoekstra, A. Boer, K. Berg, G. Sartor, R. Rubino, A. Wyner, and T. Bench-Capon, "OWL Ontology of Basic Legal Concepts (LKIF-Core)", *ESTRALLA*, p. 138, 2007.
[42] G. Venturi, and S. Montemagni, "Ontology Learning in Legal Domain: An Introduction," p. 80, 2012. Available from: http://summerschoollex.cirsfid.unibo.it/wp-content/uploads/2012/09/Ontology_Learning_in_the_legal_domain_an_introduction.pdf. [Accessed 17 July 2014]
[43] L. Hassan, "Usability Evaluation Framework for E-commerce Websites in Developing Countries," Doctoral Thesis, Loughborough University, p. 372, 2009.
[44] D. I. Zahran, H. A. Al-Nualm, M. J. Rutter, and D. Benyon, "A Comparative Approach to Web Evaluation and Website Evaluation Methods," *International Journal of Public Information Systems*, vol. 1, pp. 20–39, 2014.
[45] C. Paz-Trillo, R. Wassermann, and P. P. Braga, "An Information Retrieval Application Using Ontology" *Journal of Brazilian Computer Society*, vol. 11, no. 2, 2005. https://doi.org/10.1007/BF03192373
[46] C. D. Manning, P. Raghavan, and H. Schutze, *An Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England, p. 581, 2009.
[47] M. F. Sanchez, "Semantically Enhanced Information Retrieval: An Ontology-based Approach" Ph.D. Thesis, Escuela Politécnica Superior, Universidad Autonoma De Madrid, p. 254, 2009.
[48] P. Quaresma, and T. Goncalves, "Using Linguistic Information and Machine Learning Techniques to Identify Entities from Juridical Document," *Semantic Processing of Legal Texts, LNCS*, vol. 6036, Springer-Verlag, Berlin Heidelberg, pp. 44–59, 2010. https://doi.org/10.1007/978-3-642-12837-0_3

Esingbemi Princewill Ebietomere is currently a Lecturer, and Researcher at Services Science Laboratory of the Department of Computer Science, University of Benin, Nigeria. He obtained a Diploma certificate in Data Processing (2003), B. Sc. (2008), M. Sc. (2013) and Ph. D. (2018) in Computer Science from the University of Benin, Nigeria. His research interests include semantic web, legal informatics, ontology engineering, and information retrieval.
E-mail: princewill.ebietomere@uniben.edu

Godspower Osaretin Ekuobase is a Professor of services computing, and the Director of Services Science Laboratory at the Department of Computer Science, University of Benin, Nigeria. His research interests include web services, legal informatics, services science, and semantic web. Prof. Ekuobase obtained his B. Sc. (1998), M. Sc. (2003) and Ph. D. (2008) in Computer Science from the prestigious University of Benin, Nigeria. He has successfully supervised three Doctoral Theses, has several publications and serves as a reviewer for some reputable journals in his areas of research.
E-mail: godspower.ekuobase@uniben.edu
ORCID iD: https://orcid.org/0000-0003-3037-2529