Ontologies and the Semantic Web for Digital Investigation Tool Selection

Hayden Wimmer
Lei Chen
Tom Narock

Follow this and additional works at: https://digitalcommons.georgiasouthern.edu/information-tech-facpubs

Part of the Computer Sciences Commons

This article is brought to you for free and open access by the Information Technology, Department of at Digital Commons@Georgia Southern. It has been accepted for inclusion in Information Technology, Department of - Faculty Publications by an authorized administrator of Digital Commons@Georgia Southern. For more information, please contact digitalcommons@georgiasouthern.edu.
Ontologies and the Semantic Web for Digital Investigation Tool Selection

Hayden Wimmer
*Georgia Southern University*, hwimmer@georgiasouthern.edu

Lei Chen
*Georgia Southern University*, lchen@georgiasouthern.edu

Thomas Narock
*Notre Dame of Maryland University*, tnarock@ndm.edu

Follow this and additional works at: [https://commons.erau.edu/jdfsl](https://commons.erau.edu/jdfsl)

Part of the Computer Law Commons, and the Information Security Commons

**Recommended Citation**


Available at: [https://commons.erau.edu/jdfsl/vol13/iss3/6](https://commons.erau.edu/jdfsl/vol13/iss3/6)
ONTOLOGIES AND THE SEMANTIC WEB FOR DIGITAL INVESTIGATION TOOL SELECTION

Hayden Wimmer
Department of Information Technology
Georgia Southern University
Statesboro, GA USA
hwimmer@georgiasouthern.edu

Lei Chen
Department of Information Technology
Georgia Southern University
Statesboro, GA USA
lchen@georgiasouthern.edu

Thomas Narock
Department of Math, Physics, and Computer Studies
Notre Dame of Maryland University
Baltimore, MD USA
tnarock@ndm.edu

ABSTRACT

The nascent field of digital forensics is heavily influenced by practice. Much digital forensics research involves the use, evaluation, and categorization of the multitude of tools available to researchers and practitioners. As technology evolves at an increasingly rapid pace, the digital forensics field must constantly adapt by creating and evaluating new tools and techniques to perform forensic analysis on many disparate systems such as desktops, notebook computers, mobile devices, cloud, and personal wearable sensor devices, among many others. While researchers have attempted to use ontologies to classify the digital forensics domain on various dimensions, no ontology of digital forensic tools has been developed that defines the capabilities and relationships among the various digital forensic tools. To address this gap, this work develops an ontology using Resource Description Framework (RDF) and Ontology Web Language (OWL) which is searchable via SPARQL (an RDF query language) and catalogues common digital forensic tools. Following the concept of ontology design patterns, our ontology has a modular design to promote integration with existing ontologies. Furthermore, we progress to a semantic web application that employs reasoning in order to aid digital investigators with selecting an appropriate tool. This work serves as an important step towards building the knowledge of digital forensics tools. Additionally, this research sets the preliminary stage to bringing semantic web technology to the digital forensics domain as well as facilitates expanding the developed ontology to other tools and features, relationships, and forensic techniques.
1. INTRODUCTION

Digital forensics (DF) is a branch of forensic science that emerged in recent years. It comprises the science and practices of discovering, acquiring, preserving, analyzing, and presenting digital evidence potentially related to crime (Carrier 2003, Reith, Carr, and Gunsch 2002). There is a long list of DF tools, each of which facilitates one or more DF steps in the above process. Moreover, each DF tool has its own forensic capabilities and features and may only be compatible with certain operating system(s) and devices. Therefore, to solve a digital crime case in an efficient and accurate manner, a DF investigator must select the competent and suitable tool(s) for completing the assigned tasks.

DF investigators often encounter the following challenges when selecting the tools for their investigation. First, many investigators work for small paralegal or private investigator companies and their running budget would not allow them to use the costly DF tools, such as Encase or FTK. In such situation, they have to rely on the less expensive or free tools, such as The Sleuth Kit or Autopsy, Digital Forensics Framework and Wireshark. Second, even for investigators with abundant budget, each tool, including the class leading Encase and FTK, has its own limitations. For example, many users consider Encase provides its feature in a complex way in terms of user interface and design patterns. It has also been reported that Encase has limitation in terms of end user training and live search features (Simon 2012). As for FTK, it suffers with long response time when used for analyzing evidence across networks. Also, users may find it difficult trying to integrate FTK with third party forensic tools (Chan 2011). Third, on account of the rapid advancement of DF, new tools, new versions and new features emerge on daily basis to meet the demand, which imposes a challenge to investigators to choose the right tools and features for investigation. For example, a DF tool capable of scanning the computer Random Access Memory (RAM) on an older version of Windows for possible digital evidence may not be able to accomplish the same task on a new version of Windows. Likewise, a DF tool for running on desktop or notebook operating systems may not excel on a smart phone, a tablet device, or a virtual cloud environment. Moreover, the emergency of new targets like IoT and smart-home technology require advancement of DF technologies. Fourth, but not the least, the specifications and features (listed on official website or in documentation) of a tool may not accurately describe the tool’s real capability or in a way understood by an investigator. For example, while many tools have the feature of making forensic image of the evidence hard drive, very few of them list the tested imaging speed in specifications or data sheet. However, investigators are eager to be aware of such metrics when comparing and selecting tools. Another example is that most tools can search keywords in acquired image of evidence hard drive, but not every single tool can search keywords within the slack space, the areas in a hard drive cluster that are not used by the operating system after the end of a file.

In this work, we aim to aid the digital forensic practitioner via cataloging popular software tools and their respective features and capabilities (tested and observed by us, as end users in the field) for both mobile and desktop
devices. In order to express the relationships among the tools and capabilities, we employ an ontology coded in OWL. Our use of the emerging ontology design pattern standard facilitates extending the knowledge coded within the ontology to support the reuse and fast searches of tool features and capabilities. Our design supports integration with currently available ontologies.1

Ontologies are defined as the explicit, formal, conceptualization of a domain (Gruber 1993) and can be used to encode a common language, and set of relationships, within a domain. Ontologies stem from ancient philosophy and have been adapted into other fields such as geosciences (Narock and Fox 2015).

Following the implement of the ontology, we develop a semantic web application using open source tools to demonstrate the reasoning potential for ontologies and semantic web in the digital forensics discipline. Our application aids digital forensics investigators in selecting a DF tool and suitable alternatives via exploiting reasoning over the relationships within the ontology.

The remainder of this paper is organized as follows: section 2 presents relevant literature, section 3 details our ontology, section 4 presents the application prototype, and section 5 concludes the work.

2. LITERATURE REVIEW

2.1 Ontology and Semantic Web

Ontologies have long been used to define a domain and are increasingly becoming standardized within domains. For example, W3C has recently defined an ontology for semantic sensor networks and various medical fields, e.g. Bio Portal (2013).

The semantic web was coined as a term to describe a world wide web that was able to be processed and understood by machines and viewed as the future of the web (Berners-Lee, Hendler, and Lassila 2001). Ontology development has evolved into a collection of Semantic Web technologies that now includes software tools and methodologies. Ontologies also promote sharing, re-use, extension, and collaboration.

2.2 Related Digital Forensic Ontology

The use of ontologies in the digital forensics (DF) domain is sporadically represented in the academic literature with a wide range of applications; therefore, exploring the use of ontologies in DF is considered to have enormous potential. Schatz, Mohay, and Clark (2004) developed the FORE system for Forensics of Rich Events which stores events in an OWL ontology to represent change of state of an event. RDF, or resource description framework, has the ability to create graph structures which were employed by (Giova 2011) to strengthen the chain of custody.

There is a need to standardize the DF field which can be solved in part by the use of ontologies. Brinson, Robinson, and Rogers (2006) create an ontology of the DF domain, more specifically cyber forensics2. The authors classify the domain into technology containing subclasses hardware and software and include professions with subclasses law, academic, military, and private sector. Each subclass was further subdivided with their cyber forensics ontology classifying the domain at six levels of depth. In a similar and related effort (same academic department), Harrill and Mislan (2007) develop an ontology of small scale digital devices such as smart phones, PDAs, and their respective software. One limitation of

---

1 https://www.w3.org/TR/2016/WD-vocab-ssn-20160531/

2 Cyber Forensics is a specialization of Digital Forensics
the aforementioned ontologies is the lack of defined relationships beyond the parent child relationship thereby not leveraging the full potential of ontologies. More recently, Karie and Venter (2014) develop an updated ontology of the DF discipline and classify aspects of the domain such as software, computer, multimedia, database, device, network, telecom, internet, and wireless forensics.

Hoss and Carver (2009) recognize the potential for ontologies in DF and propose deploying ontologies to integrate and develop automated forensic analysis tools. Using ontologies to guide search was developed into a proof of concept by Slay and Schulz (2014) where a prototype application was constructed on top of an ontology and shows to improve searching and filtering the mass amounts of forensic computing data.

Continuing the nascent stages of ontologies in DF, Ćosić and Ćosić (2012) develop an ontology as a taxonomy of digital evidence in order to prevent misunderstandings of important concepts in digital evidence. The evidence collection process was coded as an ontology by Park, Cho, and Kwon (2009) where authors went beyond the basic taxonomy of parent and child classes and began constructing more meaningful relationships among the concepts in the ontology. Alzaabi (2013) also defined relationships in their proposed domain and application ontology of a smart phone environment. Furthering the work, Alzaabi, Jones, and Martin (2013) proceed into the semantic web domain discussing the role of ontologies in the semantic web and the RDF standard and further their prior smart phone ontology development.

DIALOG, or Digital Investigation Ontology, developed an independent vocabulary and worked towards encapsulating all concepts and relationships in the DF field (Kahvedžić and Kechadi 2009) and extended an application to use the ontology. DIALOG was designed to be general with specific concepts where necessary in order to facilitate its use in applications. Casey, Back, and Barnum (2015) develop an ontology called the Digital Forensic Analysis eXpression, or DFAX, to provide domain specific information to be integrated into CybOX (Cyber Observable eXpression), a language developed in collaboration between industry and academic to promote consistent capture and transfer of cyber content. DFAX leverages relationships beyond parent and child, now represented as the RDF subClassOf, by incorporating typical relationships such as “has-a”, “is-a”, as well as custom defined relationships.

3. ONTOLOGY MODEL

3.1 Development

The application of ontologies in many diverse fields has led to an evolution in ontology engineering. Initially, moderate to large domain ontologies were the norm. These ontologies attempted to model an entire application area (domain or sub-domain) within one ontology. While this may have been useful for the given application, it led to limited reuse of those ontologies. The RDF and OWL languages do not allow selective imports of semantic statements and ontology engineers were forced to reuse entire domain ontologies. This leads to two major challenges. First, universal agreement on concept hierarchies and term relationships is infeasible, if not impossible, in many scientific domains (Janowicz and Hitzler, 2012). Second, the underlying semantics of the logical languages OWL and RDF can lead to complex ontology models that are difficult for humans to understand (Blomqvist et al., 2015). As a result, ontology reuse has been limited in practice.
To combat this, the field of ontology engineering is moving toward so-called Ontology Design Patterns (ODPs). ODPs were introduced independently by Blomqvist (2005) and Gangemi (2005) and are analogous to Software Design Patterns. The intention with ODPs is to design several small reusable ontologies that each only model one particular aspect of a domain. This facilitates ontology reuse along several dimensions (Blomqvist et al., 2015). Ontology engineering with ODPs has been systematically practiced within U.S. ontology workshops since 2012 (Hitzler et al., 2015) and has seen successful reuse and integration in multiple domains (Narock et al., 2014; Krisnadhi et al., 2015).

The ontology model presented in this work is distinctive in that it leverages ODP principles as well as the recent OWL 2 standard (Hitzler et al., 2012). Specifically, we have designed an ODP modeling the features of software within the DF domain and have done so leveraging advanced modeling afforded by OWL 2. The modular nature of the resulting ODP means that it can be reused easier and better facilitate alignment and integration within the DF domain. The OWL 2 features lead to automated reasoning capabilities that make application development easier and more efficient.

3.2 Relationships

Our DF ODP is shown graphically in Figure 1. Solid blue lines indicate sub-class relationships. Dashed lines indicate a “has-a” relationship as in Software has-an Operating System it operates on. The ontology distinguishes among Operating System Capability and Operating System Compatibility (for space this is collapsed in Figure 1). The former models the operating system on which the software is installed while the latter models the operating system(s) the software is capable of analyzing. For example, a given DF tool may install on Linux, but be capable of analyzing Linux, Windows, and OS X-based systems. Of note is our use of the OWL 2 role chain in modeling Operating System Compatibility. The ontology contains the rule that if a DF software tool contains a feature that is compatible with a given OS then that software tool is compatible with the given OS. This limits the amount of data input required by ontology users and allows applications to leverage automated reasoning.

Also of note is the transitive and symmetric relationship hasSimilarFeature. This property allows analysts to relate features among DF software tools. For instance, one may want to link the feature Registry hive carving with Registry rebuilding given that they both operate on the Registry. The transitive and symmetric aspects of this property mean that machine reasoning can exploit a minimal set of hasSimilarFeatures to infer additional statements. For example, if analyst A links feature F1 to feature F2 and analyst B links feature F2 to feature F3, then reasoning can infer that F1 is related to F3 by the transitivity and also that F2 is related to F1 and F3 is related to F2, both from the symmetric nature of the property. Figure 1 illustrates the relationships among classes in the ontology. Figure 2 shows results of selecting the registry_analyzer feature when reasoning is not activated. Note, that only one connection to registry_analyzer exists. Figure 3 demonstrates the capability of reasoning as when the reasoner is activated, selecting registry_analyzer recursively returns the inferred relationships. Figure 4 shows a subset of the relationships in the ontology, specifically the relationships of registry_analyzer.
Figure 1
Illustrated relationships among classes in the ontology

Figure 2
Results without reasoning activated
Figure 3
Results with reasoning activated

Figure 4
Expanded illustration of relationships in instances of Feature
4. IMPLEMENTATION\textsuperscript{3}

4.1 SPARQL Endpoint

SPARQL, or SPARQL Protocol and RDF Query Language, is a language similar in structure to SQL that is used to interact and query RDF graphs. A SPARQL endpoint is a SPARQL Protocol for RDF compliant service that permits queries against an RDF, or similar knowledge-based ontology development languages. Oftentimes, the endpoint interacts with machines and returns results in a machine understandable format; however, humans also interact with the endpoint. Endpoints can be employed to foster ontology sharing and reuse.

A vast array of software is available for endpoint hosting. Virtuoso from Open Link software\textsuperscript{4}, is an open source, full featured, and well adopted platform for implementing SPARQL endpoints. The example in Figures 5, 6, and 7 demonstrate the basic configuration necessary to setup a SPARQL endpoint on Virtuoso. Once Virtuoso is installed on the system, it is necessary to upload the ontology via the Virtuoso Conductor. The ontology in this paper was created using OWL which is a supported format. Once logged into Virtuoso Conductor as illustrated in Figure 5 via port 8890 for default installation (e.g. http://hostname:8890), navigate to the “Linked Data” tab. Figure 6 shows the process of uploading an OWL file to Virtuoso and Figure 7 shows issuing a query in the Virtuoso web interface.

\textsuperscript{3} Code for our implementation is available upon request to the corresponding author.

\textsuperscript{4} http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/
**Figure 5**
Virtuoso Conductor Home Screen

**Figure 6**
Uploading New Linked Data from OWL File
4.2 Connecting Application to Endpoint

Once a SPARQL endpoint is established and published to the World Wide Web, users can utilize the endpoint to process SPARQL queries. This demonstration illustrates how to connect a popular graphical SPARQL Query tool, Twinkle\(^5\), to the previously configured SPARQL endpoint. Twinkle is a graphical user interface to the ARQ SPARQL engine. ARQ is an open source query engine licensed by the Apache foundation as part of the Jena Framework\(^6\) which was developed to build semantic web and linked data applications in Java. Installing Twinkle consists of downloading a compressed (zip) file and extracting it to the hard drive on a computer. Twinkle is coded in Java; therefore, the appropriate Java Runtime Environment must be installed on the computer in order to execute the program. Opening Twinkle consists of launching the .jar file (Java Archive) which is extracted in the previous step. Once Twinkle is running, one may connect to an endpoint.

\(^5\) http://www.ldodds.com/projects/twinkle/

\(^6\) https://jena.apache.org/download/index.cgi
and issue a query. Figure 8 illustrates the Twinkle application connecting to a base Uniform Resource Identifier (URI) which is configured as part of the ontology development process and can be found in the respective RDF or OWL file. Next, the Internet address of the endpoint is input as the Data URL. Finally, the user constructs the SPARQL query and retrieves results by clicking the button labeled “Run.” In this example, we issued a simple query that requests objects that are subclasses of other objects. Although not shown in this example, it is important to note that SPARQL supports sending a query to multiple endpoints. In fact, portions of a query can be answered by different endpoints and the results combined into a final result, which is then displayed to users. Thus, within a semantically enabled DF field queries can be answered in a distributed fashion.

![Twinkle interface and SPARQL query execution](image)

**Figure 8**
Twinkle interface and SPARQL query execution

### 4.3 Searching with SPARQL

We provide here two sample SPARQL queries to demonstrate the benefits of our ontology, and semantic technologies in general.
Query 1 searches a knowledge-base for all software that is compatible with the Android operating system. This query leverages the reasoning capabilities of semantic technologies as well as demonstrates advanced features of OWL 2. Our ontology includes a so-called rule chain, which in effect is an if-then type statement that a reasoner can evaluate. Within our ontology, DF software has features and those features are capable of operating on certain operating systems. We have built into our ontology the statement that “if a particular Software has a feature that operates on a given OS, then the Software operates on that OS”. As discussed in future sections, the collection of Semantic Web technologies includes reasoning software that enables automated deductions to be inferred. One way that our ontology differs from previous research is that it exploits more of these automated reasoning capabilities to lessen the amount of data that an analyst must manually import. For instance, an analyst can simply describe a software’s features and the operating systems they operate on and our application can automatically infer additional statements that must hold such as the software being compatible with those operating systems.

Query 2 demonstrates additional reasoning capabilities exploited within our ontology. We included the capability to tag features as being similar to each other. Using the OWL constructs of transivity and symmetry, we can infer all possible combinations of the similarity relationship and not burden the analyst with having to manually encode this information. This capability also adds a social and distributed component to our work. Analysts can tag features as being similar to each other, which can then aid other analysts in future searches. As mentioned previously, SPARQL supports distributing queries to multiple endpoints. In this manner, one can envision a distributed DF semantic infrastructure in which the tags of multiple DF analysts are exploited and combined.

### 4.4 Semantic Web Application for Digital Investigator Decision Support

In order to support the decision-making process of Digital Investigators, we developed a simple tool to aid in locating similar features. Oftentimes, Digital Investigators are not trained in the multitude of tools and techniques available. Even with training, the rapid pace at which technology evolves, tools and techniques change faster than investigators can learn and adapt. Based on this, it is prudent to provide software and tools to support investigators in selecting tools and techniques. The adherence to the Ontology Design Pattern (ODP) methods insures our ontology can be extended and reused. Specifically, we have provided a pattern for modeling DF software and its features. This ontology design pattern provides a reusable template that can be used by any semantic web enabled application. In addition to extension and reuse, leveraging the reasoning capabilities of semantic web-based applications facilitates automated relationship discovery and recommendations. Figure 9 illustrates a
prototype interface to our underlying Java-based application. The front-end GUI could be developed using any language that can employ Java such as a pure Java-GUI or a web-based application that makes use of Java Servlets and the aforementioned SPARQL endpoint. In the following example, a Digital Investigator is selecting a feature, registry_analyzer, and examining software that supports the feature and suggestions on related features.

![Prototype GUI for Digital Investigator Tool Selection](image)

**Figure 9**
Prototype GUI for Digital Investigator Tool Selection

Figure 10 illustrates the Java code to instantiate the class via the main method and the method to instantiate the reasoner. In the main method, first, a new instance of the class is instantiated. Next, a Jena model is instantiated, and the OWL file is read in from the file system. Next, a query is processed followed by closing the model. The method createDFModel demonstrates how to apply a reasoner to the Jena model. There are many reasoners for various applications and the decision on which reasoner fits a specific application is important in order to achieve the proper level and type of reasoning. In this instance, ONT_MEM_RULE_INF is selected to provide a transitive class-hierarchy inference.
The method `readDFOWLfromFile`, in Figure 11, surrounds the file input stream method in a try/catch block to trap an error in the event the file is not found, incorrectly formatted, etc. First, a Java IO `InputStreamReader` is instantiated which reads the OWL ontology from the local file system. The file is input into the Jena Ontology Model class. This corresponds to the `dfModel.readDFOWLfromFile()` line in the main method in Figure 10. Once this method is executed, the OWL ontology is input into a Jena model and transitive class-hierarchy has already been applied to the reasoner.

Figure 12 demonstrates the code to execute a SPARQL query against the Jena Ontology Model. First, a Java String is created with the SPARQL query as standard text. The SPARQL query in this example corresponds to the SPARQL query explained in Query 2.
After creating the string with the query, a Jena Query object is instantiated and `QueryFactory.create(queryString)` is called to generate a Jena query based on our string input. Next, a QueryExecution class is created and the query is executed via the `QueryExecutionFactory.create` method. Finally, a Jena ResultSet is created via executing the query created in the previous line. ResultSet `myData` now has the results of the query. Omitted from this example is the code that loops through `myData` and sends the output to the standard out. Figure 13 shows the output from the query. As noted in previous examples, the only relationship of registry_analyzer is registry_hive_carving. Registry_rebuilding, registry_restore, and registry_archive is automatically inferred by the Jena reasoner. This result shows that even if a Digital Investigator only knows about the registry_analyzer feature, the system will not only recommend its direct relationships but the entire set of relationships. This behavior can be configured based on the type of reasoner applied in the `createDFModel()` method.

```
private void getQuery()
{
    String queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX df: <http://www.semanticweb.org/hwimmer/ontologies/df#> " +
        "SELECT ?feature WHERE { " +
        "df:registry_analyzer df:hasSimilarFeature ?feature .}";

    Query query = QueryFactory.create(queryString);
    QueryExecution qexec = QueryExecutionFactory.create(query, _dfModel);

    ResultSet myData = qexec.execSelect();
}
```

*Figure 12*
Execution of a SPARQL Query

```
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#registry_hive_carving> )
in while
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#registry_rebuilding> )
in while
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#registry_restore> )
in while
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#registry_archive> )
```

*Figure 13*
Results from Query with Reasoner Activated
5. CONCLUSION

In this work, we seek to bring semantic web technologies to improve and support the decision process of digital investigators for digital forensic tool and feature selection. First, we employed the concept of Ontology Design Patterns (ODP) for development of a digital forensic tool ontology. The most common open and closed source software tools for digital forensics were researched and cataloged in our digital forensics ontology which was coded on the ontology web language (OWL). We created relationships among software, tools, capabilities, operating systems, mobile versus desktop, to name a few. Next, we illustrate the power of reasoning to automatically infer relationships within the ontology. We progress toward developing a semantic web-based application complete with reasoning capabilities by demonstrating the configuration of a SPARQL endpoint server. Next, we connect a popular GUI front end application to our SPARQL endpoint. We then illustrate some basic, but powerful, SPARQL queries and advance to a prototype semantic web application written in Java. The application demonstrates how reasoning can be built into an application to support the decision process of a digital investigator selecting tools for device analysis.

This work has provided an important step of illustrating how semantic web technologies can be employed in the digital forensics community in a real-world application. The ontology can be extended and re-used to improve the cataloguing of tools and techniques while our applications and demonstrations illustrate the potential of semantic web technologies in the digital forensics community.
REFERENCES


APPENDIX

Figure 1
Illustrated relationships among classes in the ontology
**Figure 2**
Results without reasoning activated

**Figure 3**
Results with reasoning activated

<table>
<thead>
<tr>
<th>Feature</th>
<th>Some Individual</th>
<th>Different Individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry analyzer</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>registry archive</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Registry_hive_carving</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Registry_rebuilding</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>registry_restore</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Report_editor</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Restore_files</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Resurrection_of_Windows_Registries_long_since_forgotten</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Seamless_access_to_all_instances_of_keys_and_values</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object property assertion</th>
<th>Data property assertion</th>
<th>Modification of object property assertion</th>
<th>Modification of data property assertion</th>
</tr>
</thead>
</table>
Figure 4
Expanded illustration of relationships in instances of Feature

Figure 5
Virtuoso Conductor Home Screen
Figure 6
Uploading New Linked Data from OWL File

Figure 7
Issuing a SPARQL Query in the Virtuoso web interface
**Figure 8**
Twinkle interface and SPARQL query execution

**Figure 9**
Prototype GUI for Digital Investigator Tool Selection
```java
public static void main(String[] args) throws Exception {

    DF dfModel = new DF();
    // Create Basic Knowledge Model
    dfModel.createDFModel();

    // Populate with your DF data (local file)
    dfModel.readDFOWLfromFile();

    dfModel.getQuery();

    // Close the model
    dfModel.close();
}

private void createDFModel() {
    _dfModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF);
}
```

Figure 10
Main Method and Creation of the Ontology Model with Reasoning Activated

```java
private void readDFOWLfromFile() {
    try {
        System.out.println("readDFOWLfromFile");
        InputStream meFile = FileManager.get().open("ont/df728_wimmer.owl");
        _dfModel.read(meFile, defaultNameSpace);

        meFile.close();
    } catch (IOException io) {
        System.out.println("File Error: "+ io.getMessage());
    }

    System.out.println("read is done");
}
```

Figure 11
Reading OWL Ontology into Jena Framework
```java
private void getQuery()
{
    String queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
                         "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
                         "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
                         "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
                         "PREFIX df: <http://www.semanticweb.org/hwimmer/ontologies/df#> " +
                         "SELECT ?feature WHERE { " +
                         "df:registry_analyzer df:hasSimilarFeature ?feature .}"
                      ;
    Query query = QueryFactory.create(queryString);
    QueryExecution qexec = QueryExecutionFactory.create(query, _dfModel);
    ResultSet myData = qexec.execSelect();
}
```

**Figure 12**
Execution of a SPARQL Query

```java
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#Registry_hive_carving> )
in while
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#Registry_rebuilding> )
in while
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#registry_restore> )
in while
( ?feature = <http://www.semanticweb.org/hwimmer/ontologies/df#registry_archive> )
```

**Figure 13**
Results from Query with Reasoner Activated