XML-hoo!: A Prototype Application for Intelligent Query of XML Documents using Domain-Specific Ontologies

Henry M. Kim
Schulich School of Business, York University, 4700 Keele St., Toronto, Ontario Canada M3J 1P3
hkim@schulich.yorku.ca

Abstract

Use of XML holds great promise for standardizing data models for realizing benefits such as lowered development costs and time for integrating inter-organizational business processes and intra-organizational knowledge management. Further benefits can be realized by formally defining common semantics in ontologies using the standardized models. Automation of business processes that require sharing knowledge represented in XML-based ontologies can then be supported. In this paper, a proof-of-concept application for using ontologies to support deduction of knowledge implicit in existing XML documents is presented. This system, called XML-hoo!, employs a customized portal user interface to answer queries about Shakespearian plays. Queries are answered by applying inference rules about these plays represented as axioms that comprise a Shakespearian ontology, composed of terminology corresponding to existing XML DTD’s. These rules are applied to plays represented in XML that are in the public domain. Hence, answers to queries such as, "Who is Romeo's father?" can be automatically deduced even though facts required for such answers are not explicitly structured in XML documents. This application demonstrates use of re-usable and sharable ontology representations to further leverage the expected proliferation of XML documents.

1. Introduction

Though many organizations have converted their documents to XML, these may not be fully utilized. Structured data is of limited value unless applications manipulate it in a way intended by the developer, and required by the user; there must be a common understanding about semantics of the structured data. Common understanding can be enforced via use of an off-the-shelf library, e.g. xCBL™ [1], or use of standardized industry-based languages, e.g. Financial products Markup Language (FpML) [2]. Both approaches can be considered top-down because they entail conformance to an external data schema and accompanying semantics. However, XML use for knowledge management systems (KMS) requires a bottom-up approach: Semantics about structured data in existing documents must be organized and applied to support knowledge extraction and discovery.

Semantics in of themselves constitute organizational knowledge, and if they are represented systematically, and consistent with XML structure of documents in the KMS' repository, they can be codified as query answering routines. This benefit can be realized by representing these semantics using ontologies. An ontology is an explicit understanding of shared understanding [3], so it represents common knowledge represented in the KMS. More expressively, it "consists of a representational vocabulary with precise definitions of the meanings of the terms of this vocabulary plus a set of formal axioms that constrain interpretation and well-formed use of these terms” [4]. Ontology axioms can be codified and applied to structured data for automated inference—i.e. query answering.

Smith and Coulter [5] discuss the use of ontologies with XML data for e-business applications; Glushko et. al. [6] posit limitations of such use, namely the impracticality of detailed formalization because of rapidly changing business needs. Through bottom up development, however, a practical approach can be taken: Domain-specific ontologies useful for focused tasks can be developed, and domain independent, generalizable ontology representations can be organized over time [7].

A promising focused task is the use of ontologies to support querying an XML repository. A review of such applications—e.g. GoXML [8], XYZFind [9], and xmlTree [10]—shows there are powerful search engines that index, classify, and store numerous XML documents on the web. However, they focus on speed and breadth of the search, rather than depth. As a result, their observed performance in answering queries is not substantially better than a regular search engine, unless the searcher is familiar with the structure of searched documents; these engines generally do not commit to representing the semantics of XML documents in order to answer the queries.
By committing to represent such semantics, is it feasible to develop an “intelligent” query application? Is it feasible to represent these semantics using ontologies? What can be learned for developing a query application for KMS? In this paper, a proof-of-concept application developed to address these questions, XML-hoo!, is presented with discussion of its ontology emphasized. An ontological commitment to represent the semantics and structure of Shakespearian plays [11], represented in XML and available in the public domain, is made. Next, excerpts of the ontology and XML-hoo! application are presented. Then concluding remarks and future work are stated.

2. XML-Hoo! Shakespearian Ontology

This ontology is developed and presented using the methodology shown below [12].

2.1. Motivating Scenario

The motivating scenario is a detailed narrative about problems faced or tasks performed by a user for which an ontology-based IS application is constructed. Here is XML-hoo!’s motivating scenario:

- An organization with an IS-based KMS is studying feasibility of migrating the system to an XML-based platform. This study entails prototyping an application for knowledge extraction/query from existing XML documents to surmise development effort and cost/benefit before converting all documents to XML. A focus group is selected to provide requirements for, and ultimately test, the prototype application. Documents existing in public domain, specifically Shakespearian plays, are chosen for the test. The reasons are the following: The focus group has read many plays, so can ably provide requirements; consistent XML element definitions exist, so an application broadly querying all Shakespearian plays can be designed; and extending the application to query about other domains—e.g. plays in general or historical writings—may be possible. The focus group then provides requirements for an application more sophisticated than their current KM system, and general query and search engines. Specifically, they want this application, XML-hoo! to answer non-trivial queries, the kind that a student studying Shakespeare may ask.

2.2. Competency Questions - Informal

Users’ requirements expressed as queries of an ontology-based application are competency questions. Since terms to pose formal queries in the ontology’s language are not yet developed, these questions are inherently informal, asked in English using vocabulary and semantics familiar to users. The following are some competency questions for the Shakespearian play ontology used for XML-hoo!.

CQ 1. Who is Romeo’s father?
CQ 2. Who said “Et tu, brute!”?
CQ 3. Where does ‘King Lear’ take place?
CQ 4. Who are all the characters in ‘Taming of the Shrew’?
CQ 5. Does Romeo utter a sonnet, and if so what does he say?

2.3. Ontology

2.3.1. Terminology - Hierarchical Model.

The terminology of the ontology comprises minimally of all terms required to formally express, but not answer, the competency questions. In turn, the expression that defines a given term is expressed using other ontology terms. Ultimately, a primitive ontology term is not defined, but mapped from a data repository. In presenting the terminology, the data schema of Shakespearian XML documents is presented pictorially as a hierarchical model, then terminologically as the ontology’s primitive terms expressed as predicates. Next, key terms and relationships in the informal competency questions are identified and integrated into pictorial (ER) and terminological (predicate) models.
attributes or child’s unique identifier. In an ontology, predicates that are not formally defined are called primitive terms because they are populated through assertions, not by inference. Therefore, relationships between terminal nodes in the XML data structure correspond to primitive terms of the ontology.

2.3.3. Primitive Terms - Predicate Model. Here is a description of primitive term variables. Variable numbers match those in Fig. 3.

1. P: Name of the play; value of Play⇒Title
2. S: Subtitle of play; value of Play⇒Play Subtitle
3. Scd: One scene description for the play; value of Play⇒Scene Description
4. Pa: A character list, one descriptive name for section wherein all characters are introduced; value of Play⇒Persona⇒Title
5. Pe1: Character description set of all characters individually introduced; value of Play⇒Persona⇒Persona
6. G: Character description for each grouping of characters; value of Play⇒Persona⇒Persona
7. Pe2: Character description set of all characters introduced within a given group; value of Play⇒Persona⇒Persona
8. A: Title of an act within the play; value of Play⇒Act⇒Title
9. Std1: One stage direction for each act; value of Play⇒Act⇒Stage Direction
10. Sc: Title of scene within an act; value of Play⇒Act⇒Scene⇒Title
11. Std2: One stage direction for each scene; value of Play⇒Act⇒Scene⇒Stage Direction
12. Sp: Speaker of a speech; value of Play⇒Act⇒Scene⇒Speech⇒Speech
13. L: A line in a speech; value of Play⇒Act⇒Scene⇒Speech⇒Line
14. Px: Free text, which will not be represented using the ontology; value of Play⇒P

PT-1. play_has_act(P, A) e.g. play_has_act("The Tragedy of Julius Caesar", "ACT I").
PT-2. play_has_subtitle(P, S) e.g. play_has_subtitle("The Tragedy of Julius Caesar", "JULIUS CAESAR").
PT-3. play_has_scene_description(P, Scd) e.g. play_has_scene_description("The Tragedy of Julius Caesar", "Dramatis Personae").
PT-4. play_has_character_list(P, Pa) e.g. play_has_character_list("The Tragedy of Julius Caesar", "Dramatis Personae").
PT-5. character_list_has_character_description(Pa, Pe1)

1. ⇒ denotes has
e.g. character_list_has_character_description('Dramatis Personae', 'JULIUS CAESAR').

PT-6. character_list_has_group(Pa,Gd)
e.g. character_list_has_group('Dramatis Personae', 'tribunes').

PT-7. group_has_character_description(Gd,Pe2)
e.g. group_has_character_description('Dramatis Personae', 'tribunes', 'FLAVIUS').

PT-8. act_has_stage_direction(A,Std1)

PT-9. act_has_scene(A,Sc)
e.g. act_has_scene('ACT I', 'SCENE I. Rome. A street').

PT-10. scene_has_stage_direction(Sc,Std2)

PT-11. scene_has_speech(Sc,Sp,L1)
L1: First line of a speech. Though not explicitly an element, this attribute in combination with Sp is used to uniquely identify a speech
e.g. scene_has_speech('SCENE I. Rome. A street', 'FLAVIUS', 'Hence! home, you idle creatures get you home').

PT-12. speech_has_line(Sp,L1,L)
e.g. speech_has_line('FLAVIUS', 'Hence! home, you idle creatures get you home', 'Is this a holiday? what! know you not').

Each primitive term can be populated by a query on XML documents. Take the question, “Given that 'Tragedy of Julius Caesar' is the title of the play, what is the play’s subtitle?”. Using First-Order Logic, the question is expressed using primitive terms as the following axiom to prove: \( \exists S \ play_has_subtitle('The Tragedy of Julius Caesar', S) \).

Using the query language XML-QL, the question is expressed as follows, and returns the value, \( \$st = 'JULIUS CAESAR' \).

WHERE <PLAY>
  <TITLE>Tragedy of Julius Caesar</>
  <PLAYSUBT>$st</PLAYSUBT>
</PLAY>

CONSTRUCT <results>
  <PLAYSUBT>$st</PLAYSUBT>
</results>

Though not explicitly structured using XML elements, there is an observed format for introducing characters, which applies with few exceptions. For instance, the value of the <PERSONA> element always starts with the character’s name, and may be proceeded by combinations of pseudonym, qualifiers, and statements of relationship with other characters.

\[ \text{<PERSONA} \] ::= \text{<description set>}
\]
\[ \text{<description set>} ::= \text{<character name>}
\]
\[ \text{<character name>} ::= \text{[<character pseudonym> ]}
\]
\[ \text{<character pseudonym>} ::= \text{[<adjective> | <adverb> ]}
\]
\[ \text{<adjective> | <adverb>} ::= \text{<character title>}
\]
\[ \text{<character title>} ::= \text{[<preposition> | <article> ]}
\]

\[ \text{<relationship of character>} ::= \text{[<conjunction> | <relation title> ]}
\]
\[ \text{<conjunction>} ::= \text{[<relation noun> | <relation preposition> ]}
\]

\[ \text{<relation noun> ::= <relation preposition>}
\]

1 Formats in bold italic are primitive formats—those not defined in terms of other formats—significant for the ontology. Though other formats like <adverb> and <preposition> are primitives also, they markup extraneous information, which need not be represented using the ontology.
Figure 4: Format for Persona and Group Descriptions, expressed in BNF notation

Obviously, the implementation to parse values within an element is not trivial and is an issue for future work. Nevertheless, assuming parsing capability, following are the primitive terms, which express relationships between <PERSONA> or <GRPDESCR> and the primitive formats such as <character name> and <relation noun>.

\[ \text{Pe: } \text{Pe}_1 \text{ (individual character description) or } \text{Pe}_2 \text{ (description of individual characters described in a group description); value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Pd: } \text{Description for one character; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{D: } \text{Pd or Gd (group description; value for } \langle \text{GRPDESCR} \rangle \text{ element)} \]

\[ \text{C: } \text{Just the name of the character; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Ps: } \text{Pseudonym of the character; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Qt: } \text{A qualifying title of a character, e.g. 'King'; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Lq: } \text{A location that qualifies a character's title, e.g. 'King of Denmark'; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Cr: } \text{A character who is referenced when describing another character; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Rn: } \text{The noun describing the nature of the relation between a character and Cr, e.g. father; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{Rp: } \text{Relation preposition that qualifies Rn, e.g. 'of' in 'father-of'; value for } \langle \text{PERSONA} \rangle \text{ element} \]

\[ \text{PT-13. } \text{character_description_has_primitive_description}_{\text{set}}(\text{Pe}, \text{Pd}) \]

\[ \text{PT-14. } \text{primitive_description_set_has_character}(\text{Pd}, \text{C}) \]

\[ \text{PT-15. } \text{primitive_description_set_has_pseudonym}(\text{Pe}, \text{Ps}) \]

\[ \text{PT-16. } \text{description_has_qualifying_title}(\text{D}, \text{Qt}) \]

\[ \text{PT-17. } \text{description_has_location_qualifier}(\text{D}, \text{Qt}, \text{Lq}) \]

\[ \text{PT-18. } \text{description_has_relationship}(\text{D}, \text{Rn}, \text{Rp}, \text{Cr}) \]

2.3.4. Ontology Data and Predicate Models. From the informal competency questions, the following key words are isolated: who, son, said/utter, where... take place, characters, sonnet, and what... say. Obviously, some of these words can be easily defined using the primitive terms as:

\[ \text{Pred-1. } \text{character}(\text{C}) \]

\[ \text{Pred-2. } \text{location}(\text{Lo}) \]

\[ \text{Pred-3. } \text{speaker_starts_speech_with}(\text{Sp}, \text{L}) \]

These in turn are used to define ontology terms such as the following:

\[ \text{Pred-4. } \text{play_has_character}(\text{P}, \text{C}) \]

\[ \text{Pred-5. } \text{play_has_location}(\text{P}, \text{Lo}) \]

\[ \text{Pred-6. } \text{has_father}(\text{C}_1, \text{C}_2) \]

\[ \text{Pred-7. } \text{speaker_says}(\text{Sp}, \text{L}) \]

\[ \text{Pred-8. } \text{speaks_sonnet}(\text{Sp}, \text{L}_1, \text{L}_s) \]

A sonnet—its contents being a list Ls—is spoken by speaker Sp and starts with line L1.

2.3.5. Formal Competency Questions. Competency questions can now be posed, and formally expressed in First-Order Logic. Given a set of axioms in an ontology (Tontology), and a set of instance objects and relations...
2.3.6. Axioms

A competency question Q is a First-Order sentence that can be entailed or is consistent with these sets; i.e., Tontology ∪ Tground |= Q or Tontology ∪ Tground |= ¬Q.

CQ-1. Which character is the father of the Romeo character?

∀C has_father(Romeo',C).

CQ 2. Which speaker says the line, “Et tu, brute!”?

∃C speaker_says(C,'Et tu, brute!).

CQ 3. What is the location for the play ‘King Lear’?

∃Lo play_has_location('King Lear',Lo).

CQ 4. Which are all the characters in the play ‘Taming of the Shrew’?

∃L∀C [ C ∈ L → play_has_character('Taming of the Shrew',C) ].

CQ 5. Does Romeo speak a sonnet, and what are the lines that comprise it?

∃Lo1∃Lo2 speaks_sonnet('Romeo',Lo,Lo1).

Each question corresponds to an axiom. To prove it, ontology axioms defining and constraining use of terms comprising the question axiom must exist.

2.3.6. Axioms

To answer CQ-1, the following predicates are formally defined.

Defn-1. character(C)

The first part of a primitive description set for one character is the character’s name.

∀C∃Pd [ primitive_description_set_has_character(Pd,C) ↔ character(C) ].

Defn-2. play_has_character_description(P,Pe)

A character description Pe either is in a list of individual character descriptions, or contained within a list of group descriptions.

∀Pe∀P∃Pa [ play_has_character_list(P,Pa) ∧ ( character_list_has_character_description(Pa,Pe) ∨ ∃Gd ( character_list_has_group(Pa,Gd) ∧ group_has_character_description(Gd,Pe) ) ) ] ↔ play_has_character_description(P,Pe).

Defn-3. play_has_character(P,C)

A character name C is the first part of a primitive description set describing one character, which is part of a list of character descriptions

∀P∀C∃Pe∃Pd [ play_has_character_description(P,Pe) ∧ character_description_has_primitive_description_set(Pe,Pd) ∧ primitive_description_set_has_character(Pd,C) ] ↔ play_has_character(P,C).

Defn-4. character_has_pseudonym(C,Ps)

A primitive description set describing one character can have both the character’s name and pseudonym used.

∀C∀ Ps∃Pd [ primitive_description_set_has_character(Pd,C) ∧ primitive_description_set_has_pseudonym(Pd,Ps) ↔ character_has_pseudonym(C,Ps) ].

Defn-5. a) related_characters(C1,Rn,Rp,C2)

C1 has a relationship, expressed as relation noun(Rn)+preposition(Rp), with C2, if: C1 is explicitly stated as related to C2 or C2’s pseudonym; C1 is a character introduced individually, or is any of the characters in a group that has a relationship to C2; and C1 and C2 are characters in the same play.

∀C1∀C2∀Rn∀Rp∃D [ ( description_has_relationship(D,Rn,Rp,C1) ∨ character_has_pseudonym(C1,Cr) ) ] ∧ ( primitive_description_set_has_character(D,C1) ) ∨ ∃Pe3Pd [ group_has_character_description(D,Pe) ∧ character_description_has_primitive_description_set(Pe,Pd) ] ∨ ∃P [ play_has_character(P,C1) ∧ play_has_character(P,C2) ] ) → related_characters(C1,Rn,Rp,C2).

Defn-6. b) related_characters(C1,Rn,Rp,C2)

C1 has a relationship, expressed as relation noun(Rn)+preposition(Rp), with C2 if: C1 is a pseudonym for a character whose relationship with C2 can be inferred; C2 is a pseudonym for a character whose relationship with C1 can be inferred, or C1 and C2 are pseudonyms for characters whose relationship with each other can be inferred.

∀C1∀C2∀Rn∀Rp∃C1,R2 [ related_characters(C1,Rn,Rp,C1) ] ∨ [ related_characters(C2,Rn,Rp,C2) ] ∨ related_characters(C1,Rn,Rp,C2) ∨ related_characters(C2,Rn,Rp,C1) ∨ related_characters(C1,Rn,Rp,C2) ) → related_characters(C1,Rn,Rp,C2).

Defn-7. a) may_be_related_characters(C1,Rn,Rp,C2)

C1 may have a relationship, expressed as relation noun(Rn)+preposition(Rp), with C2, if: C1 and C2’s relationship (Rn+Rp) cannot be inferred for sure; C1 is explicitly stated as related to C2’s qualifying title or location qualifier; C2 is a character introduced individually, or is any of the characters in a group; C1 is a character introduced individually, or is any of the characters in a group that has a relationship to C2, and C1 and C2 are characters in the same play.
In the next section, these axioms are applied to answer competency questions.

2.4. Demonstration of Competency

Obviously, many such relationship terms can be defined, e.g. has_mother, has_uncle, or additional definitions has_father. Also possible familial relationships can be defined using may_be_related_characters.

Definitions for answering CQ-2 and CQ-3 are straightforward, so are not presented. The predicate play_has_character has been defined, so CQ-4 can be answered. It is difficult to precisely answer CQ-5 because defining when lines rhyme in a sonnet—consisting of three quatrains and a couplet with a rhyme scheme of abab cdcd efef gg—is complicated. Furthermore, existing Shakespearian XML documents do not represent paragraph separations, so quatrains and couplets cannot be defined without modifying the documents. The following axiom at least disqualifies a speech that is obviously not a sonnet.

Cons-1. If a speech does not have 14 lines, it cannot be a sonnet.

\[ \forall S \forall L \exists L_1 \forall L \in L \land \text{speeches}\_\text{has}\_\text{liner}(S, L_1, L) \land n(L) = 14 \implies \neg \text{speeches}\_\text{sonnet}(S, L_1, L) \]

where \( n(X) \) is a function that returns the \# of elements in a list \( X \).
implicit relationship can be inferred using the axioms: 

- applying Defn-2 to (i) & (ii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague'). 
  - applying Defn-2 to (i) & (ii), infer 
    \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-3 to (xvi), (xvii) & (xxviii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-3 to (xvi), (xvii) & (xxviii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-3 to (xvi), (xvii) & (xxviii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

Figure 7: Relevant Primitive Term Instances

So, the following competency question is answered.

**CQ-1.** Which character is the son of the Montague character?

\( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-2 to (i) & (ii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-2 to (i) & (ii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-2 to (i) & (ii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

- applying Defn-2 to (i) & (ii), infer 
  \( \exists C \) has_father(\'Romeo',C). returns has_father(\'Romeo', 'Montague').

3. **XML-hoo!**

**XML-hoo!** is presented to the user via a web browser. The user can perform: 1) guided search through a topic classification tree, 2) keyword search using a traditional search engine, or 3) a menu-driven conceptual search of Shakespearian plays. Since the first two are capabilities provided by existing XML portals, this paper will discuss the third.

The user is presented with the following screen in which context sensitive menus are presented:

![XML-hoo! Conceptual Search Interface](image)

**Figure 9: Answering CQ-6**

How **XML-hoo!** answers competency questions is presented below.

A more interesting question is, "Who are the kinsmen to Escalus?"**, being no direct relationship reference to 'Escalus' is stated in character introductions. Rather, there are references to 'prince,' which describes him. This implicit relationship can be inferred using the axioms:

**CQ 6.** Who is possibly Escalus' kinsman?

\( \exists C \) may_be_related_characters(C,kinsman,'to','Escalus'). returns 

\( \exists C \) may_be_related_characters(C,kinsman,'to','Escalus').
The pre-defined relationships in the menu correspond to predicates defined in the ontology. The diagram below explains how a user query is answered.

**Figure 11: XML-hoo! Main Systems Architecture, and Query Answer Process**

In (1), the Control Module, a Java program, parses the query represented from the user interface, and translates and expresses it as a competency question in the ontology implementation language.

In (2), the control module then passes the question to the Ontology Query Engine, which interacts with the Ontology Repository—the Prolog programming environment is used to implement both components—to prove the competency question axiom (3). In (4), a set of primitive term queries that need to be answered are then passed back to the Control Module. In (5), the control module then translates each query in the XML query language, XML-QL, and passes them to the XML Query Engine, which repeatedly queries XML documents in the repository (6). In (7), answers to XML queries are returned to the control module, which returns a set of populated primitive terms to the Ontology Query Engine, which then proves the competency question—i.e., repeats steps (2)-(4). In (8), the Control Module formats the answer, if it exists, or an error statement, and returns that to the User Interface.

The rationale for the systems architecture is consistent with ontology use: Sharability and re-usability. The application can scale up to a variety of users querying about different domains and from different repository sources by de-coupling the Control Module, XML Query Engine and Repository, and Ontology Query Engine and Repository, albeit at the expense of inefficiency for focused users and a centralized repository source.

**4. Concluding Remarks and Future Work**

The development of this ontology, and the XML-hoo! application based on it, provide the following evidence to support using domain specific ontologies to represent semantics for XML documents in a knowledge management system:

- The capability of the Shakespearian ontology to support inference of facts not explicitly structured in XML demonstrates that an ontology-based approach to query answering is a natural complementary function for an XML data repository. Familiar relationships are not structured in XML. Plus, nowhere is it explicitly nor parametrically represented that Montague is Romeo’s father. Yet, this can be reasoned in XML-hoo!.
- The thorough example presented in this paper provide useful instructions for an ontology building endeavor to complement an XML repository. In fact, the example is novel insofar as it is an application of a traditional ontological engineering methodology to develop XML-based ontologies.
- The XML-hoo! application serves as a first pass reference for any endeavor to develop a knowledge management application using XML-based ontologies.

The crux of the XML-hoo! application is the Shakespearian play ontology. Its development can be described as follows:

- An ontological engineering methodology is followed to state the motivating scenario and competency questions.
- The hierarchical structure for a common set of XML documents, namely Shakespearian plays, is translated to develop primitive terms—ontology predicates that are populated by look-ups into an XML document, rather than inferred using formal definitions.
- Additional structure within an element is discerned; e.g., there is a format for character introductions that holds with few exceptions, which applies to the `<PERSONA>` element. This structure is translated to develop more primitive terms.
- Ontology predicates are identified from competency questions and ensured for consistency with primitive terms. This is sufficient to express the competency questions formally in the language of the ontology using predicates.
- Axioms that define meanings of predicates or constrain their interpretation are developed. By applying ontology axioms to populated primitive terms, answers to competency questions are inferred.

This work can be extended several ways. There is an immediate opportunity to formalize and automate manipulation of format (grammar) within an XML element not further structured. This is inherently difficult, and achieving perfect manipulation (parsing) is unlikely. However, the systematic perspective of an ontologist can be applied to make simplifying assumptions in specifying grammar and restricting the domain of discourse described using that grammar. It is believed that a tractable parsing system can result, and a prototype for XML-hoo! is currently in development.
Beyond formally representing semantics to support automatic inference, ontologies are desirable for use because its representations are sharable and re-usable. So not only can ontologies be used to more richly answer queries about data structured in XML, they are inherently re-usable as more documents are added to a repository. From this prototype, it appears that the Shakespearian play ontology’s definitions are indeed applicable for other types of plays and literature, and can be de-coupled from specific structural definitions. Future work will endeavor to provide stronger evidence of this. Though the definitions of primitive terms may need to be changed as similar but different DTD’s are added to a repository, higher level definitions may be re-used with little modification. As several iterations of the ontological engineering methodology are applied, a structuring of the ontologies for a repository will emerge: General ontologies will be defined in terms of specific ontologies’ representations. Whereas XML is used to structure data, ontologies can then serve to structure knowledge composed from data. This supports the role of ontology-based languages to proceed XML towards Berners-Lee’s Semantic Web [13].

5. Acknowledgements

The author expresses gratitude to the students in his Business Information Systems Analysis, who assisted in detailing the ontology and user interface.

6. References