Representing TEI Documents in the CLASSIC Knowledge Representation System

Nancy Ide, Tim McGraw, Chris Welty
Vassar College
Department of Computer Science
Poughkeepsie, New York 12604-0252
{ide,timcgraw,weltyc}@cs.vassar.edu

1. Introduction

The development of the Text Encoding Initiative (TEI) Guidelines enables the encoding of a wide variety of textual phenomena to any desired level of fine-grainedness and complexity, relevant to a broad range of applications and scholary interests. The ability to encode complex phenomena has, in turn, created a demand for adequate means to manipulate the text once it has been marked up according to the user's interests and needs. One obvious and immediate need for users of the TEI scheme is a flexible means to query and retrieve from an encoded text, which does not require deep knowledge of the structure of the text by the user. There has been some work in this area [1], although so far most systems require that the user know the structure of the document as defined by the DTD.

Beyond the need to query and retrieve based on tags which exist in a TEI document, a means to manipulate and query *classes* of objects is also desirable. The TEI DTD uses SGML entity definitions to create "classes" of elements and attributes, in particular, for groups of elements with common structural properties (e.g., all elements that can appear between paragraphs), groups of attributes which apply to certain classes of elements (e.g., attributes for pointer elements), etc. In addition to grouping together elements and attributes with common structural properties, the definition of such classes recognizes common *semantic* properties among elements and attributes. However, the SGML entity definition mechanism provides only for string substitution within the DTD itself, thereby enabling easy reference to these classes in later element definitions; the common semantic properties that are implicit in the classification scheme are lost for the purposes of retrieval and document manipulation. Obviously, a means to refer to and manipulate classes of elements and attributes in a query and retrieval system would provide substantial additional power for the user.

We are experimenting with the representation of a DTD and associated documents (i.e., documents conformant to the DTD) in a knowledge representation (KR) system, in order to provide more sophisticated query and retrieval from TEI documents than current systems

provide. We are using CLASSIC, a frame-based representation system developed at AT&T Bell Laboratories [2]. Like many KR systems, CLASSIC enables the definition of structured concepts/frames, their organization into taxonomies, the creation and manipulation of individual instances of such concepts, and inference such as inheritance, relation transitivity, inverses, etc. In addition, CLASSIC provides for the key inferences of subsumption and classification [4]. By representing a document as an individual instance of a hierarchy of concepts derived from the DTD, and by allowing the creation of additional user-defined concepts and relations, sophisticated query and retrieval operations can be performed. This paper briefly describes the CLASSIC system, the representation of a DTD and a document conforming to that DTD in CLASSIC, and provides an overview of the kind of query and retrieval that can be performed.

2. The CLASSIC Knowledge Representation System

A fairly readable and more complete description of CLASSIC can be found elsewhere [2]. We present here a brief overview of the facilities CLASSIC provides that can be used to represent DTDs and documents.

Historically, CLASSIC is a descendant of KL-ONE, and is in the family of languages known as *description logics*. Such languages attempt to deal with terminological reasoning and automatic classification [5].

There are three kinds of formal objects in CLASSIC:

- *concepts* Concepts are descriptions of classes of objects. In general, a concept description includes only the *sufficient* properties of that class, and this enables subsumption reasoning. For example, if we define AUTHOR as a concept which is both a kind of a PERSON *and* has written something, then any instance of PERSON which has written something is *automatically* classified as an AUTHOR.

- *individuals* Individuals are instances of concepts. This is typically the data in a knowledge-base. For example, PERSON is a concept, and CHRIS may be an individual of PERSON.

- *roles* A role is a relationship between individuals, and is also used to store simple attributes of individuals. For example, the individual CHRIS may have a role called NAME, which is filled with the string "Chris"--a simple attribute. If the individual CHRIS has a role called EMPLOYEE-OF, which is filled with the individual VASSAR (an individual of the concept COLLEGE), then EMPLOYEE-OF is considered a relationship between the two individuals CHRIS and VASSAR.

An ontology in CLASSIC contains the concept and role definitions, which describe what things

can be.

CLASSIC provides for the following kinds of inference:

- *inheritance* When an individual is asserted to be an instance of a certain concept, the individual is assumed to be described by that concept, and therefore it inherits all the properties described in the concept and all the properties defined in the parents of that concept. This is a fairly standard mechanism provided by most KR and object-oriented languages.

- *propagation /transitivity* CLASSIC provides for specifying rules that can be used to add more relationships to individuals. For example, we can define a rule that says if A is PART-OF B and B is PART-OF C, then A is PART-OF C.

- *inverses* Roles can be defined as inverses of each other. For example, if we define the role HAS-PART to be the inverse of the role PART-OF, then if we say A is PART-OF B, then we automatically infer that B HAS-PART A. In the example above, this would be combined with the transitivity of PART-OF to give us C HAS-PART B *and* C HAS-PART A.

- *role hierarchies* Roles can be defined to have parents, which are more general roles. When an individual has a value for a role which has a parent, the individual is inferred to have the same value for the parent role. For example, if we define that the role ENGINE-OF has a parent role PART-OF, and we say that individual A has the value B for its ENGINE-OF role (i.e. A ENGINE-OF B), then we infer A PART-OF B.

- *classification/subsumption* CLASSIC can automatically determine membership of an individual in a certain class even when not specified. This is an immensely powerful tool. For example, let us define the concept FACULTY as follows: (and PERSON (all employee-of COLLEGE)). This means that the concept FACULTY is a subconcept (subclass) of PERSON, *and* that the EMPLOYEE-OF role is filled with an individual of the concept COLLEGE. Let us say that we next create an individual of PERSON, CHRIS, with a simple value in its NAME role, the string "Chris". In CLASSIC , this would be specified as (and PERSON (fills NAME "Chris")). Next we create an individual of the concept COLLEGE called VASSAR. At some later time, we add information to the individual CHRIS, indicating that the value for the EMPLOYEE-OF role is VASSAR. CLASSIC will automatically infer that CHRIS is now an individual of (is subsumed by) the concept FACULTY.

3. DTDs in CLASSIC

We have developed an algorithm to derive an ontology from the definitions in a DTD that reflects the hierarchy of elements defined by that DTD. Each element in the DTD is represented in CLASSIC as a *concept*. An additional concept, PCDATA, is defined in order to provide a covering concept for text data. The roles *contains*, *first*, and *next*, are also defined in the ontology, and are constrained based on the content models in the DTD. Each of these roles has an inverse, *contained-in*, *first-of*, and *previous*, respectively. For simplicitly, this algorithm has been tested by applying it to the TEI-conformant CES DTD defined in the Corpus Encoding Standard (CES) [3].

The roles in the derived ontology are not meant to capture the complete syntactic meaning of the DTD. The *contains* role, for example, simply indicates that one element can be structurally contained in another. The *first* and *next* roles are used to represent the order. For example, the element definition:

```
<!ELEMENT  keywords      - -  (term+ | list)  >
```

would correspond to the following concept definitions:

```
keywords :: (and dtd-element
            (all contains (or term list))
            (all first (or term list)))

term     :: (and dtd-element
            (all next term))
```

This is an over-simplication, which is actually incorrect in a number of details, but the example illustrates that the syntax specified in the DTD is loosely captured in the ontology as containment and sequencing restrictions. In this example, individuals of KEYWORD are restricted to having CONTAINS roles filled with individuals of TERM or LIST, and having FIRST roles with the same restriction. Note that this is not necessarily a connection between the concept KEYWORDS and the concept TERM (or LIST), but is rather a restriction that is passed on to individuals of KEYWORDS. Attributes of elements are captured as roles with simple string values (with one exception, noted in the next section). The restrictions on the attribute values specified in the DTD are not represented. For example, the following attribute definition in a DTD:

```
<!ATTLIST div  %a.text;
                  complete       (y | n)  >
```

specifies that the values of the COMPLETE attribute must be either Y or N. The ontology generated for this will simply say:

```
div :: (all complete string)
```

(in addition to any other structural information specified for this element) which says the complete role must have a string for a value, and does not restrict the string to "Y" and "N".

Note that our system does not do error checking on documents; there are many tools to test DTD conformancy, so it is not necessary to capture these restrictions.

4. User-defined classes

CLASSIC allows for the definition of classes of concepts, one of the most basic capabilities of any KR language. Therefore, once a set of concepts corresponding to elements is created from a DTD, additional classes can be defined which reflect additional structural and semantic properties of and among these concepts. For example, the elements P, QUOTE, LIST, NOTE, etc. could be defined to be elements belonging to a class of **paragraph-like objects** (which would be appropriate according to the definitions in the CES DTD), reflecting their structural similarity (i.e., all appear just below the DIV level but not one inside the other) as well as their semantic similarity (they are all large linguistic units comprised of one or more sentences). Similarly, the concepts corresponding to the elements NAME, AUTHOR, etc. can all be defined as a class of concepts with the superclass **person-name**, reflecting their semantic similarity. "Person-name", in turn, can be a subclass of a broader class of concepts called, say, **name**, which would include, in addition to **person-name**, other names such as PLACENAME, ORGNAME, etc. Application-dependent relations can be defined as well: for example, a linguistic application might define a class of concepts called **token** (corresponding to unbreakable tokens for linguistic analysis), including, NAME, DATE, NUM, TIME, etc.

Once defined as members of a class, groups of elements can be queried, manipulated, etc. as a group, rather than as a list of individuals. In addition, properties can be associated with any class which are then automatically "inherited" by all members of that class; for example, if the class **name** is defined as having sub-parts FIRSTNAME and LASTNAME, then all members of the class **person-name** are defined to have those sub-parts. Queries can also exploit inherited properties.

The *type* attribute is treated specially when generating an ontology from a DTD. In the CES DTD, the *type* attribute is used consistently to denote subtypes. For example, the element **name** has a type attribute which has values such as PERSON, ORGANIZATION, etc. Some elements in the DTD specify a closed list of values for the TYPE attribute (the **measure** element in the CES DTD defines the possible values of *type* to be one of weight, length, count, area, volume, temperature, currency), but most do not. If a range of values is specified, each of these becomes a subconcept of the concept representing the element. If not, the user may create subconcepts manually, or may wait for these subconcepts to be created automatically when documents are processed (see below).

Roles may also be associated with any CLASSIC concept, enabling the creation of more complex relations. For example, a concept **person** can be established with a role *name*; this role can be filled by any individual object (element) which is a **person-name**, such as NAME, AUTHOR, etc. Another role, *affiliation*, could also be associated with **person** and filled by an individual of an appropriate concept/element such as **organization**.

It is important to note that we are here defining semantic relationships, as opposed to the syntactic relationships defined by the DTD. For example, in a marked-up document in which a P element occurs within a BODY element, it is tempting to refer to the BODY element as a parent of the P element, since this is true for the parse tree representing the document. However, here we refer to this kind of structural relationship using the *contains* role, and we would therefore say the BODY element *contains* the P element.

In our terminology, one element is the parent of another only when it is more general--analogous to a superclass in an object-oriented language. However, there is no way to express this kind of parent relationship in SGML. The CES DTD and the TEI DTD both attempt to use entities to create "classes" of elements and attributes, but because entities are only a string-replacement mechanism, there is no means by which class membership can be recognized or manipulated by typical SGML-aware software. The ability to manipulate and query on the basis of class membership is one of the most important and promising aspects of the CLASSIC representation for SGML documents.

5. Documents in CLASSIC

Documents conformant to a DTD represented in CLASSIC comprise *individuals* which instantiate the concepts in the ontology derived from the DTD. Documents entered into CLASSIC are assumed to be syntactically well-formed with respect to their corresponding DTD. For example, the following simple HTML marked-up document:

```
<HTML
<TITLEExample</TITLE
<BODY<PThis is a simple example.</P</BODY
</HTML
```

would correspond to the following individuals in CLASSIC :

```
HTML-1  :: (and HTML
                (fills contains TITLE-2 BODY-3)
                (fills first TITLE-2))

TITLE-2 ::  (and TITLE
               (fills contains CDATA-4)
               (fills first CDATA-4)
               (fills next BODY-3))
```

```
BODY-3  ::  (and BODY
                (fills contains P-5)
                (fills first P-5))

P-5     :: (and P
                (fills contains CDATA-6)
                (fills first CDATA-6))

CDATA-4 :: (and CDATA
               (fills text "Example"))

CDATA-6 :: (and CDATA
               (fills text "This is a simple example."))
```

The name of each individual is generated automatically, and consists of the name of the most specific parent concept and a number to make the name a unique symbol. These names have no meaning to CLASSIC; they are simply place holders.

6. Using CLASSIC to Query Documents

There are a number of manipulations that can be performed once documents have been stored in CLASSIC. Complex path-tracing searches can be specified, similar in style to an SQL query of a relational database.

The main difference between a KR system like CLASSIC and a relational database is the ability to specify inferences that eliminate the need for a user to be completely familiar with the DTD. That is, knowledge of what the structural relationships in the document actually *mean* can be specified in the ontology and used to make the job of searching for information easier for the user.

For example, a marked-up document conforming to the CES DTD might contain a NAME element, with type=PERSON, within the BODY part of a document. When this element is encountered, our system will read the type attribute and, if this is the first time that type has been encountered, it will create a new concept with NAME as its parent, and then create an individual of that new concept that represents the tagged text.

Now suppose we decide that "any person-name occuring within the BODY of a document is a character." We create a new role called *character,* which is attached only to documents, and we can specify that the *character* role of a document is filled automatically with values according to the rule "any person-name occuring within the BODY of a document is a character" (the syntax for specifying such a rule is beyond the scope of this article). Subsequently, retrieval can be performed on the basis of this defined relationship; for example, all characters can be retrieved, or all characters with a certain set of characteristics, etc.--even if there is no tag for character.

Note also that such queries would differentiate names applying to persons vs. names applying to places, organizations, etc., as well as names appearing in the body and names appearing elsewhere (for example, in the header). This overcomes a problem that arises due to the lack of scoping rules in SGML: for example, the element NAME may have a very different semantics depending on whether it appears in the header or in the body of the text, but SGML provides no way to differentiate instances of a given tag on the basis of context. [6]

The representation of DTDs and associated documents in CLASSIC provides considerable facility for document query. Details are excluded due to space restrictions, but consider in particular the following capabilities:

- selection of or reference to particular portions of a document, enabling queries which, for example, access information in the header only, in titles only, in running prose only, etc.;
- reference to classes rather than specific instances of a class, enabling, for example, queries on anything which is a **person-name**, regardless of whether the actual tag is PERSNAME, AUTHOR, etc. This is particularly powerful because it avoids complex queries involving boolean expressions (e.g., "persname OR author OR (name WITH-ATT 'person') IN body", etc.), and the user is not required to be familiar with the DTD in order to formulate a query;
- access to information not explicitly available from the structure of the document, but which is inferred by CLASSIC. For example, a **person** is a class of individuals that exist separate from the DTD-based ontology. When a person's name is read in as an author of a document, we can infer that person is an **author**.

Such facility offers promise for querying document databases such as those being developed for the Word Wide Web, etc [7]. For example, queries such as the following are possible:

- all **authors** where the last name is "Smith"
- all **persons** where the last name is "Smith"
- all technical articles written by **persons** with affiliation "Vassar"
- all technical articles written by **authors** with last name "Smith" and affiliation "Vassar"
- all **authors** of technical articles
- all articles containing a mention of a **person** with name "Smith" (either as an author or mentioned in the article, etc.)
- all **persons** in the header

- all **persons** with no affiliation
- all **dates** that appear within paragraph-like elements only (i.e., not in the header, headings, titles, etc.)

7. Conclusion

The representation of SGML documents in a knowledge representation system such as CLASSIC offers the potential to provide considerably more powerful query and retrieval capabilities than have previously been available. In particular, it will enable the manipulation of and access to elements within documents on the basis of semantic rather than purely syntactic (structural) properties. Classes of elements can be accessed, and knowledge of the DTD is not essential for constructing queries. Further, CLASSIC's inferencing capabilities can provide access to information that is not directly retrievable from the document structure, upon which all current systems rely.

The representation of SGML documents in CLASSIC may also have repercussions for DTD design. So far, DTD design has been largely unprincipled, and wide variations in the kinds of information represented by elements, attributes, and tag content often occur, even within the same DTD. However, the formal representation of elements, attributes, and content as CLASSIC objects demands consistency in their use within the DTD. The development of a set of principles for DTD design is a desideratum among the encoding community; we are looking into the ways in which formatlization of DTDs in CLASSIC can contribute to this development.

At the same time, the use of a system such as CLASSIC allows for greater flexibility in tagging text. For example, for names of people, the encoder can use the general NAME tag--or even more generally, RS (referring string)--or provide a very precise encoding using PERSNAME with FIRSTNAME, LASTNAME, etc. elements inside. Once represented in CLASSIC, these objects can be both recognized as members of the class **person-name** and accessed and manipulated as such. This frees the encoder to choose an encoding for each name that is appropriate; there is no need for absolute consistency to enable the semantic identity of the two elements to be recognized. More generally, it allows precise tag semantics to be instantiated in a system external to the encoded text.

Acknowledgements

The present research has been partially funded by US NSF RUI grant IRI-9413451, and by AT&T Bell Laboratories.

References

[1] See, for example, Blake, G.E., Consens, M., Davis, I.J., Kilpelainen, P., Kuikka, E, Larson, P.-A., Snider, T., Tompa, F.W. Text/Relational database management systems: Overview and proposed SQL extensions. Available at http://solo.uwaterloo.ca/trdbms/. See also Harie, S., Ide, N., Le Maitre, J., Murisasco, E., Véronis, J. (1996). SgmlQL - An SGML Query Language. Proceedings of SGML'96, p. 127.

[2] Brachman, R, Borgida, A., McGuinness, D., and Resnick, L. The CLASSIC Knowledge Representation System. In *Proceedings of the 11th IJCAI*. August, 1989.

[3] Ide, N., Priest-Dorman, G., Véronis, J. (1996). Corpus Encoding Standard. Documentation and DTDs available at http://www.cs.vassar.edu/CES/.

[4] Brachman, R. What is-a is and isn't. *IEEE Computer*. October, 1983. pp 30-36.

[5] Minsky, M. A Framework for Representing Knowledge. *Mind Design.* MIT Press. Pp. 95-128. 1981.

[6] The CES attempts to solve this problem by renaming elements such as NAME, TITLE, AUTHOR, etc. when they appear in the header (e.g., H.NAME, H.TITLE, etc.). However, this solution is clearly ad hocand only complicates the information the use is required to remember if retrieval is based solely on the specifications in the DTD.

[6] Welty, Chris. Intelligent Assistance for Navigating the Web. *Proceedings of The 1996 Florida AI Research Symposium*. May, 1996.