. . . Back to The SHOE Home Page

SHOE

Simple
HTML
Ontology
Extensions

SHOE 1.01

Proposed Specification
Sean Luke and Jeff Heflin
SHOE Project
April 28, 2000

Latest version of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec.html
Version 1.0 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec1.0.html
Version 0.993 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.993.html
Version 0.992 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.992.html
Version 0.991 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.991.html
Version 0.99 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.99.html
Version 0.98 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.98.html
Version 0.97 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.97.html
Version 0.96 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.96.html
Version 0.95 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.95.html
Version 0.90 of this document: http://www.cs.umd.edu/projects/plus/SHOE/spec0.9.html

Table Of Contents

1 Introduction

This specification describes SHOE, an extension to HTML which provides a way to incorporate machine-readable semantic knowledge in HTML or other World-Wide Web documents. SHOE can also be used with XML documents, and this process is discussed in Appendix B. The rest of this specification describes:

- A hierarchical classification mechanism for HTML documents (and optionally non-HTML documents) or subsections of HTML documents.
- A mechanism for specifying relationships between classified elements and other classified elements or specific kinds of data (numbers, dates, etc.)
- A simple way to specify ontologies containing rules that define valid classifications, relationships, and inferred rules.

The intent of this specification is to make it possible for user-agents, robots, etc., to gather truly meaningful information about web pages and documents, enabling significantly better search mechanisms and knowledge-gathering.

The general way one goes about this is as follows:

- First, define an ontology describing valid classifications of web objects, and valid relationships between web objects and other web objects or data. This ontology may borrow from other ontologies.
- Annotate HTML pages to describe themselves, other pages, or subsections of themselves, as having attributes as described in one or more ontologies.

We're playing a bit fast-and-loose with the term *ontology* here. In this specification, "ontology" simply means an ISA hierarchy of classes/categories, plus a set of atomic relations between these categories, and a set of inferential rules in the form of simplified horn clauses. Categories inherit relations defined for parent categories.

User agents following this specification should be aware that assertions made by HTML pages are not *facts,* but *claims*. I.e., if element x claims that element y is related with relation r to element z, then the user-agent should not be entering `r(y,z)` into its database (i.e., "Now I know that y is related to z with the relationship r!!"). Instead, it should be entering something along the lines of `r(x,y,z)` into its database (i.e., "x is claiming that y is related to z with relationship r."). This is an important distinction: it's perfectly fine for HTML pages out there to be making completely false claims; one shouldn't simply accept them as truth. For similar reasons, HTML pages can only make *assertions*, not *retractions*.

Readers of earlier versions of this specification may notice that there have been some significant changes to the syntax of SHOE. These changes were necessary to make SHOE an SGML application (see Appendix A: SHOE and SGML). However, the semantics of the language are relatively unchanged.

1.1 Extensions to HTML

SHOE adds the following tags to the HTML standard:

To make possible ontology declarations, SHOE adds `ONTOLOGY`, `/ONTOLOGY`, `USE-ONTOLOGY`, `DEF-CATEGORY`, `DEF-RELATION`, `/DEF-RELATION`, `DEF-ARG`, `DEF-RENAME`, `DEF-CONSTANT`, `DEF-TYPE`, `DEF-INFERENCE`, `/DEF-INFERENCE`, `INF-IF`, `/INF-IF`, `INF-THEN`, `/INF-THEN`,

`COMPARISON`, `/COMPARISON`, `CATEGORY`, `RELATION`, `/RELATION`, and `ARG`.

To make possible semantic markup of HTML pages, SHOE reuses some of the tags from above and adds `INSTANCE` and `/INSTANCE`. The other tags used for markup are `USE-ONTOLOGY`, `CATEGORY`, `RELATION`, `/RELATION`, and `ARG`. Additionally, SHOE declares the `META HTTP-EQUIV` tag "`SHOE`".

1.2 Indicating a SHOE-conformant Document

SHOE-conformant HTML documents must declare themselves SHOE documents and provide versioning information. To be conformant with this version of the specification, SHOE documents must have the following text in the *HEAD* section of the document:

```
<META HTTP-EQUIV="SHOE"
          CONTENT="VERSION=1.0">
```

In addition to indicating the version of SHOE, the version number indicates which version of the base ontology is applicable. There is a one-to-one correspondence between a version of SHOE and a version of the base ontology.

2 Terms

Terms not described here may be found in the HTML 2.0 specification.

Category (Class)
: These two terms are used interchangeably, as are "categorization" and "classification". A category is an element under which HTML page instances or subinstances can be classified. Category names are element names, and may be prefixed. Categories may have parent categories. Categories define inheritance: if an instance is classified under a category, keys classified with this category may fill argument positions in relations defined for that category or any of its parent (or ancestor) categories. Multiple inheritance is valid. See "Name" below for rules about the construction of category names.

Data
: Data which can be placed in an argument of a relationship. Data may either be an instance (see "Instance" below) or one of the types defined in an ontology. The base ontology includes the following types:

  Strings (`STRING`)
  : HTML String Literals, as defined in the HTML 2.0 specification.

  Numbers (`NUMBER`)
  : Floating-point numerical constants. Knowledge-agents should be able to read common floating-point numbers like `2, 2.0, -1.432e+4`, etc. Numbers may be of the form,
    `[+|-] ((digit+ ['.' digit+]) | ('.' digit+)) [((e|E)[+|-]digit+)]`
    This rather liberal form should be readable by the C scanf command if it is read as a float.

  Dates (`DATE`)
  : Date/Timestamps following RFC 1123, as shown in section 3.3.1 of the HTTP/1.0 specification. . More information on internet date formatting can be found in Section 5 of RFC 822 .

Booleans (`TRUTH`)
HTML String Literals of the form `YES` or `NO`, case-insensitive.

Data types defined in an ontology other than the base ontology are called arbitrary data types. The format of data stored under these types is implementation-specific.

When an ontology refers to a data type from another ontology (including the base ontology), a prefix chain must be used. See "Prefix" below for a description of prefix chains. However, empty prefix chains indicate the base ontology and thus `.STRING`, `.NUMBER`, `.DATE`, and `.TRUTH` can be used to specify data types for any relation, regardless of the current ontology.

Element
A category, relationship, or type. Elements within an ontology share the same namespace. See "Name" below for rules about constructing element names.

Instance
A data object which may be classified under zero or more categories, and included as an argument to relationships (along with other forms of data). *Subinstances* are instances that are declared within another instance. *Constant instances* are declared within ontologies. Instances form the most common data entities in databases built up from this specification.

Instance Key
A string which uniquely defines an instance. It is up to you to decide on the keys for your documents. The base key of a SHOE-conformant document is a single absolute URL of the document. Typically, these documents will have one instance whose key is the base key. For example, `http://www.cs.umd.edu` is a valid key for one instance within a document located at that URL.

Many documents will have multiple instances, each of which must have a unique key. To create keys for additional instances within a document, add a unique pound-suffix, such as `#MyDog`, to the base key. For example, `http://www.cs.umd.edu#MyDog` is a valid key for an additional instance located at `http://www.cs.umd.edu`. It is good style for this key to correspond with an actual anchor in the document. Some documents may not have an instance whose key is the base key; instead all instances within it have a pound-suffix key. In fact, this approach is recommended if multiple instances exist at the outermost level (they are not enclosed within other instances).

The unique key of a non-SHOE-conformant document is defined to be one particular absolute URL of the document, chosen for the document by a SHOE-conformant document which references it.

Strings used as keys are subject to a few restrictions. Keys must begin with a letter or number and may not contain whitespace. They are also case-sensitive. The SHOE spec reserves the key "me" and any capitalized form of it. "me" (under any capitalized form) may be used as an argument of a claim to refer to the enclosing data instance actually making the claim. The SHOE spec reserves any keys beginning with "!" for referencing *constant instances*. See section 3.7 for more information on how to form such keys.

Name
A name identifies a category, relation or type. Within a given ontology all names must be unique. A name may contain only letters, numbers, and hyphens; no whitespace is allowed. Names are case-sensitive. The following conventions are recommended for naming objects:

- *Category* - First letter is capitalized; rest is mixed case
- *Relation* - First letter is lowercase; rest is mixed case
- *Type* - All letters are capitalized

Ontology

As defined in this specification, a description of valid classifications for HTML instances and valid relationships between instances and elements. The *base ontology* is the ultimate ancestor of all other ontologies. All ontologies directly or indirectly extend the base ontology. All documents that conform to this version of the SHOE Specification use version 1.0 of the base ontology.

Ontology Identifier

A string which uniquely defines an ontology. Ontology identifiers are different from *keys* in that they do not uniquely define *instances* but rather the ontologies which the instances may use. Different versions of an ontology may have the same identifier so long as they have different version numbers. Identifiers must begin with a letter or number, and may not contain whitespace. Identifiers are case-sensitive.

Prefix

A small string attached with a period at the beginning of an instance, category, relation, or type name. For example, `cs` is a prefix in `cs.junk`. Prefixes may also be attached to already-prefixed elements, forming a *prefix chain.* For example, `foo.bar.cs` is a prefix chain for `foo.bar.cs.junk`. A prefix indicates the ontology from which the element (or prefixed element) following it is defined. If the prefix chain consists of just a single period, the element is assumed to come from the base ontology.

Relation (Relationship)

An element which defines a relationship between instances and other instances or data. Relation names are element names, and may be prefixed. A relation is between zero or more elements called *arguments*; if a relation is defined for some set of arguments, this permits SHOE documents to declare this relation among instances of those arguments. Arguments are explicitly ordered, so each has a numbered position (the first is argument 1). Many relations are binary (have exactly two arguments). A binary relation's *domain* is argument 1 of the relation. A relation's *range* (the element the relation is "to") is argument 2 of the relation. SHOE reserves for future use any relation names containing a character other than a letter, a number, or a hyphen.

Rule

A formal rule in an ontology defining valid classifications (categories) or valid relationships that can be asserted.

Version (Version Number)

A string which describes the version of an ontology. Versions are case-sensitive, and may contain only letters, digits, or hyphens.

3 Declaring Ontologies

Ontology declarations may appear at the top level within the body of an HTML document (i.e., they may not be enclosed by any other tags within the `BODY`). HTML tags may not occur within an ontology declaration.

For each tag described in this section, there is a specification of the syntax in a `system font`. In these specifications, *italics* indicate that the author can supply any string that meets the naming conventions of that component and brackets ('[' or ']') enclose optional portions of the syntax.

Arbitrary white space can be included between attributes within a tag and between tags.

3.1 Declaring An Ontology Definition

An HTML document may contain any number of ontology definitions. Each ontology definition should use a unique identifier. Ontology definitions are accompanied with a version number. If an ontology completely subsumes previous versions of the same ontology (it contains all the rules defined in those versions), it may declare itself to be backward-compatible with those versions. To begin an ontology definition, use:

```
<ONTOLOGY ID="ontology-identifier"
          VERSION="version"
         [BACKWARD-COMPATIBLE-WITH="version list"]
         [DESCRIPTION="text"]
         [DECLARATORS="list-of-declaring-instances"]>
```

ID (mandatory)
: The ontology's unique identifier.

VERSION (mandatory)
: The ontology's version.

BACKWARD-COMPATIBLE-WITH
: A whitespace-delimited list of previous versions which this ontology subsumes.

DESCRIPTION
: A short, human-readable description of the purpose of the ontology.

DECLARATORS
: A whitespace-delimited list of keys for instances the ontology has associated with itself. Ordinarily an ontology cannot *declare* relationships or categorizations, only define the rules that govern such declarations. This mechanism allow an ontology to state that one or more specific instances are making important standard declarations associated with the ontology. It is important to note that it is the instances themselves that make the declarations, not the ontology (See section 4 to make instances).

To end an ontology definition, use:

```
</ONTOLOGY>
```

All rules and extensions in an ontology must appear between the beginning and ending declarations. Ontologies may not be nested or overlap.

3.2 Extending An Existing Ontology

An ontology may use elements from one or more existing ontologies in its own rules. The ontology is said to *extend* the existing ontologies. To distinguish between those elements and its own elements, an ontology must provide a unique prefix for each ontology it extends. This will be prefixed to elements borrowed from each particular ontology whenever they are referred to. To declare that an ontology is extending another ontology, use:

```
<USE-ONTOLOGY ID="ontology-identifier"
          VERSION="version"
          PREFIX="prefix"
         [URL="URL"]>
```

ID (mandatory)
: The extended ontology's unique identifier.

VERSION (mandatory)
: The extended ontology's version.

PREFIX (mandatory)
: The prefix you are assigning the extended ontology. All categories, relations, and types from the extended ontology which are used in your ontology must be prefixed with this prefix. Within an ontology or an instance, a prefix must be different from all prefixes declared with other `<USE-ONTOLOGY>` tags.

URL
: A URL that points to a document which contains the extended ontology.

3.3 Defining Classification Rules

Inside an ontology definition, an ontology may define various new categories which instances can belong to. Categories should descend from one or more parent categories. To define a new category, or to add new parent categories for a category, use:

```
<DEF-CATEGORY NAME="category-name"
         [ISA="parent-category-list"]
         [DESCRIPTION="text"]
         [SHORT="text"]>
```

NAME (mandatory)
: The newly declared category, or the one being given more parent categories. Newly declared categories should be distinct from all other categories, relationships and types declared in the ontology. Category names may not include whitespace. It is recommended that the name be of mixed capitalization (for example, "LaunchVehicleNASA"). SHOE reserves for future use all category names that begin with a character other than a number or a letter.

ISA
: A whitespace-delimited list of categories to define as parent categories of this category.

DESCRIPTION
: A short, human-readable description of the category's semantics.

SHORT
: A phrase which an agent may use to display the category to a user in a more understandable fashion than the category's name. In English ontologies, `SHORT` should be a singular or mass noun, lower-case unless it is a proper noun. For example, the category "LaunchVehicleNASA" might have `SHORT="rocket"`.

A particular category should not be defined more than once within an ontology's declaration.

3.4 Defining Relationship Rules

Inside an ontology definition, an ontology may define various new valid relationships between category instances or between category instances and data. To define a relationship, use:

```
<DEF-RELATION NAME="relation-name"
```

```
        [DESCRIPTION="text"]>
        [SHORT="text"]>
   argument-list
</DEF-RELATION>
```

NAME (mandatory)
: The newly defined relationship name. This should be distinct from all other categories, relationships, and types declared in the ontology. It is recommended that this name be of mixed capitalization with the first letter uncapitalized (for example, "wentToSchoolAt"). SHOE reserves for future use all relation names that begin with a character other than a number or a letter.

DESCRIPTION
: A short, human-readable description of the relation and its arguments.

SHORT
: A phrase which an agent may use to display the relation to a user in a more understandable fashion than the relation's name. In English ontologies, `SHORT` should be a verb for singular subjects, lower-case, and in the format that makes some sense when appearing after the first argument but before the remaining arguments. For example, the relation "nameOf" might have `SHORT="is named"`.

argument-list
: A (possibly empty) set of arguments, one listed after another. Each argument is defined by:

```
<DEF-ARG POS=(positive-integer | FROM | TO)
         TYPE="data-type"
         [SHORT="text"]>
```

POS (mandatory)
: The position of the argument being defined. One of two formats should be followed. N-ary relations must use a positive integer to specify which argument is being defined. For the first argument, `POS` must equal `1` and each successive argument should be assigned the next greatest integer. It is illegal to re-use an integer or skip a number. Alternatively, binary relations (those consisting of exactly two arguments) can use the `FROM` and `TO` values to define the positions of the first and second argument, respectively. If a relation defines an argument with `POS=FROM` then it must also define exactly one other argument with `POS=TO`. The reverse is also true.

TYPE (mandatory)
: The type of the argument. This should be a single declared category, ontology-defined type, or one of the following types from the base ontology: `.STRING, .NUMBER, .DATE, .TRUTH.`

SHORT
: A description of each argument in the relation. This should be a lower-case singular or mass noun, or a (correctly cased) proper noun.

A particular named relationship should not be defined more than once within an ontology's declaration.

3.5 Renaming Rules

To reduce the number of prefixes, an ontology may rename any element (along with its prefix chain) to a simpler name, so long as this name is not used by another element in the ontology.

Elements include categories, relations, types and constant instances. For example, an ontology could rename the category `cs.junk.foo.person` to simply `person`, so long as `person` is not defined elsewhere in the ontology.

To rename a category, relation, constant instance, or type, use:

```
<DEF-RENAME FROM="element-name"
          TO="new-element-name">
```

FROM (mandatory)
: The element's old name.

TO (mandatory)
: The element's new name. This name is subject to the standard rules for constructing names.

3.6 Defining Inference Rules

An ontology may declare one or more inferences which may be automatically made based on the relationship and category declarations found in marked-up SHOE text.

A SHOE inference clause is similar to a *Horn clause* in that it consists of a *body* of one or more subclauses describing claims entities might make, and a *head* consisting of one or more subclauses describing a claim that may be inferred if all the claims in the body do turn out to be made.

Subclause Rules. Both the head and the body of an inference clause *must* contain at least one subclause; inference clauses with an empty head or body are incorrect and may be ignored. A subclause in the head may either be a relation declaration or a category declaration. A subclause in the body may be either a relation declaration, a category declaration, or a comparison declaration as described below. Comparison declarations may not be made in the head. Comparison declarations permit inferences to include equals/not-equals/greater-than/less-than relationships between elements in subclauses.

Constants and Variables. The data in a head or body subclause may be either *constants* or *variables*. Constant data must be matched exactly as it is. Variables may be matched (*bound*) to any constant data interned by a SHOE agent, so long as variables of the same name in subclauses within the same inference clause always bind to the same value, and fill argument positions of the same type. Variables are *case-insensitive*: the variable V matches with the variable v.

Variable Rules . There are two variable rules, intended to eliminate ambiguities and excess computational complexity in the inference clauses:

- The head of the inference clause may not contain variables that do not appear somewhere in the body of the clause.
- If there is more than one variable in the body, then each variable must appear at least once in a relationship-type (`<RELATION>`) subclause within the body along with another variable in such a way that one could trace a path from any variable to any other variable through a series of relationships (in other words, the relationship graph for variables in the body is *connected*). Comparison declarations (other than "equal") and category declarations do not count as part of this relationship graph.

Inference clauses that do not meet these constraints are incorrect and may be ignored.

A SHOE inference has the form:

```
<DEF-INFERENCE
          [DESCRIPTION="text"]>
   <INF-IF>
      body-subclauses
   </INF-IF>
   <INF-THEN>
      head-subclauses
   </INF-THEN>
</DEF-INFERENCE>
```

DESCRIPTION
: A short, human-readable description of the inference's semantics.

body-subclauses
: One or more body subclauses, as defined in section 3.6.1, below.

head-subclauses
: One or more head subclauses, as defined in section 3.6.2, below.

3.6.1 Defining an Inference Body Subclause

An inference may have one or more body subclauses; this set is enclosed by the `<INF-IF>` and `</INF-IF>` tags. Each body subclause is either a category declaration, a relationship declaration, or a comparison declaration. Body subclauses indicate claims that must be actually made by SHOE documents for the inferences in the head subclause to take place.

To define that a body subclause is a category declaration, use:

```
<CATEGORY NAME="prefixed.category"
          FOR="instance"
          [[USAGE=]("VAR" | "CONST")]>
```

NAME (mandatory)
: A category with full prefix chains showing a path through extended ontologies back to the ontology in which it was defined.

FOR (mandatory)
: Contains the key of an actual instance or a variable to be bound to an instance which has been declared to belong to this category.

USAGE
: Indicates if *instance* refers to the key of an actual instance (`CONST`) or to a variable (`VAR`). A variable will be bound to an instance which has been declared to belong to this category. If no `USAGE` is specified, it defaults to `CONST`. Note that even when a value is specified, the `USAGE=` is optional.

To define that a body subclause is a relationship declaration, use:

```
<RELATION NAME="prefixed.relationship">
```

```
   argument-list
</RELATION>
```

NAME (mandatory)
: A relationship with full prefix chains showing a path through used and extended ontologies back to the ontology in which it was defined.

argument-list
: The set of arguments for the relation, specified one after another. A relationship must have the same number of arguments as specified in the ontology in which it was defined. If a relation subclause has missing arguments or contains more arguments than the relation definition specifies, the inference clause is incorrect and may be ignored. To define each argument, use:

```
<ARG POS=(positive-integer | FROM | TO)
         VALUE="key"
         [[USAGE=]("VAR" | "CONST")]>
```

POS (mandatory)
: The position of the argument being defined. One of two formats should be followed. Binary relations (those consisting of exactly two arguments) can use the `FROM` and `TO` values to define the positions of the first and second argument, respectively. N-ary relations must use a positive integer to specify which argument is being defined. Each successive argument should be assigned the next greatest integer. It is illegal to re-use an integer or skip a number.

VALUE (mandatory)
: Declares the element in argument position indicated by the `POS` subtag. For example, `<ARG POS=7 VALUE="George">` declares that the constant "George" is argument 7 in the relation. This object must be of the type declared for that particular argument position.

USAGE
: Declares whether the element for this argument is a variable (`VAR`) or constant (`CONST`). For example, `<ARG POS=7 VAR VALUE=VAR1>` declares that the variable VAR1 is argument 7 in the relation. If no value is specified, `USAGE` defaults to `CONST`. Note that even when a value is specified, the `USAGE=` is optional.

To define that a body subclause is a comparison declaration, use:

```
<COMPARISON OP="special.declaration">
   argument-1
   argument-2
</COMPARISON>
```

OP (mandatory)
: One of the special declaration key words: `equal, notEqual, greaterThan, greaterThanOrEqual, lessThanOrEqual,` or `lessThan`. These are all *binary* special declarations, and indicate that argument 1 is equal, not equal to, greater than, or less than argument 2. For strings, instance keys, category names, and relation names, these declarations assume case-sensitivity, and greaterThan/lessThan have no meaning. For dates, assume that earlier dates are "less than" later dates. For truths, assume that NO is "less than" YES.

argument-1

The first operand of the comparison. The syntax and semantics is the same as for a relation argument.

argument-2

The second operand of the comparison. The syntax and semantics is the same as for a relation argument.

A comparison must have exactly two arguments. It is incorrect for an ontology to declare comparison declaration subclauses that have any other number of arguments; if this happens, the inference clause is incorrect and may be ignored.

3.6.2 Defining an Inference Head Subclause

An inference may have one or more head subclauses, the set of which is enclosed by the `<INF-THEN>` and `</INF-THEN>` tags. Each head subclause is either a category declaration or a relationship declaration. A head subclause indicates a claim that may be inferred if all the claims defined in the body subclauses are met.

To define that a head subclause is a category declaration, use the `<CATEGORY>` tag. The syntax and semantics are the same as those for defining a category subclause in the body of the inference as described above.

To define that a head subclause is a relationship declaration, use the `<RELATION>` tag. The syntax and semantics are the same as those for defining a relation subclause in the body of the inference as described above.

3.7 Declaring a Constant Instance

When declaring an ontology definition, the DECLARATORS subtag might be used to declare standard instances like "Red" which are subclasses of the ontology-defined class `Color` and are used in the `colorOf` relationship. However, the key for "Red" may then turn out to be something like "`http://www.cs.umd.edu/ontology-associated-instance.html#Red`". This is not very nice for common, very standard instances (like colors). To remedy this, SHOE allows ontologies themselves to declare instances, using:

```
<DEF-CONSTANT NAME="constant-instance-name"
         [CATEGORY="prefixed-category-name"]>
```

NAME (mandatory)

The name of the constant instance. This name is subject to the standard naming rules described in Section 2.

CATEGORY

A single category (including full prefix chains) under which the instance is to be categorized.

The key for this instance is defined as a "!" followed by the constant instance name plus its prefix chain. For example, if an ontology defined "Red" as a constant instance, and some instance uses this ontology with the "cs" prefix, then the instance can reference "Red" with the key "!cs.Red". Note that this means that there may be more than one key referencing the same constant instance, depending on the particular path of prefixes chosen. All such keys should resolve to the same instance.

If the ontology wants to do anything additional to the instance (such as making more categorizations or relationships for it), it must rely on the DECLARATORS mechanism to do this.

If you're trying to define a new data type that requires special interpretation of the data string, and does not have a small, predefined set of possible values, it's probably better to declare an arbitrary data type to do this.

3.8 Defining Arbitrary Data Types

The base ontology of SHOE defines the global data types `STRING`, `NUMBER`, `DATE`, and `TRUTH`. Each of these data types interprets data inside the HTML string differently. For example, if "2345" is a STRING, it's interpreted as the string "2345". If it's a NUMBER, it is interpreted as the integer 2345. It is possible for an ontology to declare its own data types; how to *interpret* data stored under an ontology-defined type is implementation dependent. SHOE reserves for future use any type names containing a character other than a letter, a number, or a hyphen.

Unlike SHOE global types which can have empty prefix chains (that is, there is no string before the period), ontology-defined types must be referenced just as ontology-defined categories are: with the prefix chain back to the ontology that defines the type. To define a type, use

```
<DEF-TYPE NAME="type-name"
          [DESCRIPTION="text"]
          [SHORT="text"]>
```

NAME (mandatory)
: The newly defined type name. This should be distinct from all other categories, relationships, and types declared in the ontology. It is recommended that this name be in ALL CAPS. SHOE reserves for future use all type names that contain a character other than a number, a letter, or a hyphen.

DESCRIPTION
: A short, human-readable description of the type's semantics.

SHORT
: A phrase which an agent may use to display the type to a user a more understandable fashion than the category's name. In English ontologies, `SHORT` should be a singular or mass noun, lower-case unless it is a proper noun.

If you want to declare a data type consisting of a choice of one or more predefined elements (for example, buildings on a university campus or the primary colors red/green/blue), it's better not to use an arbitrary data type to but instead use one or more constant instances.

4 Marking Up HTML Documents Using Ontologies

4.1 Declaring an Instance

Instance declarations may appear at the top level within the body of an HTML document (i.e., they may not be enclosed by any other tags within the `BODY`). HTML tags may not occur within an instance declaration. Any declarations made within an instance are considered to be claims by it. To declare the start of an instance, use:

```
<INSTANCE KEY="key-value"
          [DELEGATE-TO="instance-list]>
```

KEY (mandatory)
: The unique key for the instance.

DELEGATE-TO
: Specifies the keys of instances that are permitted to make declarations on behalf of this instance. This should be a whitespace-delimited list of valid keys.

Typically, the delegated instance will declare a special *subinstance* of the same key name as the permitting instance's key. Agents should consider all claims made within that subinstance as if they were made by the permitting instance itself. This might be done to consolidate claims for a web site into a single document, perhaps, or to eliminate a large number of claims from slowing down the download time of a document. If the delegated instance does not in fact declare this special subinstance, then delegating declarative power is simply a pointer to an agent to look elsewhere for relevant SHOE knowledge.

Delegated subinstances should be considered proxies for the delegating instance, and any subinstances contained within a delegated subinstance should be considered proxies for subinstances within the delegating instance. An agent can guess that a subinstance is a delegated subinstance instead of an ordinary subinstance by examining its key. If the key is not formed from the base key of the enclosing instance, it's more than likely a delegated subinstance for some instance in another document. However, an agent should not take this at face value, but should check the delegating instance first to make certain that the delegation is valid.

To mark the end of an instance, use:

```
</INSTANCE>
```

All category and relationship declarations must appear between the start and end tags of an instance. An instance may also declare other instances ("subinstances") *nested* within it. The syntax for declaring a subinstance is the same as for declaring an instance.

4.2 Declaring Ontology Usage

Before you can classify instances or establish relationships between them, you'll need to define exactly which ontologies these classifications and relations are derived from, and associate with each of these ontologies some prefix unique to that ontology. An instance may declare that it is using as many ontologies as it likes, as long as each of these ontologies has a unique prefix. To declare that an instance and all its subinstances use a particular ontology, use:

```
<USE-ONTOLOGY ID="ontology-identifier"
          VERSION="version"
          PREFIX="prefix"
          [URL="URL"]>
```

ID (mandatory)
: The ontology's unique identifier.

VERSION (mandatory)
: The ontology's version.

PREFIX (mandatory)
: The prefix you are assigning the ontology. All categories and relations from this ontology which are used in this instance must be prefixed with this prefix. The prefix *must* be different from all prefixes declared with other `<USE-ONTOLOGY>` tags within this instance or any enclosing instances.

URL
: A URL that points to a document which contains the used ontology.

Note that the syntax for `USE-ONTOLOGY` is the same when it appears inside of an instance as when it appears inside an ontology definition.

4.3 Declaring Categories

Instances may be *classified*, that is, they may be declared to belong to one or more categories in an ontology, using the `CATEGORY` tag:

```
<CATEGORY NAME="prefixed.category"
         [FOR="key"]>
```

NAME (mandatory)
: A category with full prefix chains showing a path through used and extended ontologies back to the ontology in which it was defined.

FOR
: Contains the key of the instance which is being declared to belong the category. If `FOR` is not declared, then the key is assumed to be that of the enclosing instance. If `FOR` is declared, then it provides the key.

4.4 Declaring Relationships

Instances may declare relationships with elements. To declare relationships between elements, use:

```
<RELATION NAME="prefixed.relationship">
   argument-list
</RELATION>
```

NAME (mandatory)
: A relationship with full prefix chains showing a path through used and extended ontologies back to the ontology in which it was defined.

argument-list
: A set of argument declarations, one after another. To declare each argument, use:

```
<ARG POS=(positive-integer | FROM | TO)
         VALUE="key">
```

POS (mandatory)
: The position of the argument being defined. A positive integer indicates that this argument fits that position in the list of arguments defined for the relation. `FROM` is synonymous with `1`. `TO` is synonymous with `2`. Rules for use of arguments are specified below.

VALUE (mandatory)
: Declares the element in argument position indicated by the `POS` subtag. For example, `<ARG POS=7 VALUE="George">` declares that the constant "George" is argument 7 in the relation. This object must be of the type defined for that particular argument position.

Relation declarations take two forms, the "binary" form and the "general" form.

The "binary" form is used only for binary relationships. In this form, `POS` can be `FROM` or `TO`. One of these arguments may be omitted if the corresponding element type is `INSTANCE`. In this case, the key for the element is assumed to be that of the enclosing instance. If the tag *is* declared, and the type of the tag's argument is an instance, then it provides the key.

The "general" form may be used for relationships of any number of arguments (including binary relationships). In this form, `POS` is set to `1, 2, 3, 4, ...`, etc. It is incorrect for an instance to declare relationships with a different number of arguments than were defined for that relation in the ontology. If the number of arguments does not correspond, the relationship is incorrect and may be ignored. Note that this is different from the assumptions in the "binary" form.

Regardless of form, it is incorrect for two or more arguments to refer to the same `POS`. If a relation declares arguments of this nature, it may be ignored. If no arguments are specified with the <ARG> tag, then the "general" form is assumed, with zero arguments declared.

The reason for two different forms is that while the "general" form is necessary to describe all possible relationships, the large majority of relationships are *binary* relationships between the *claimant* and some other instance or data. Hence, the "binary" format allows the claimant to declare binary relationships in a more natural format ("to" and "from") and not have to include himself in the claim when it's clear from the context. The "binary" form makes relations feel more like slots in the claimant "object" than ordinary predicate relationships.

Appendix A: SHOE and SGML

SHOE has been designed to be an application of the Standard Generalized Markup Language (SGML). SGML is an ISO standard for document markup, and HTML is now specified as an SGML application. All SGML applications require a Document Type Definition (DTD). The DTD specifies what tags can be used in a document and the structure of the relationship of these tags to one another. SHOE builds upon the HTML 3.2 DTD by redefining the `block` entity to include the elements ONTOLOGY and INSTANCE, and then defining the corresponding sub-elements. The formal SHOE DTD is located at http://www.cs.umd.edu/projects/plus/SHOE/shoe.dtd. A copy of the DTD is below:

```
<!-- DTD for SHOE -->
<!-- Last Mod: 1/1/98 -->

<!ENTITY % shoe.content "ONTOLOGY | INSTANCE" >
```

```
<!-- The following three entity declarations are used to override
     the HTML content model for blocks, so that an ONTOLOGY or
     INSTANCE can appear anywhere a block can. Typically this is
     as a top level element in the BODY of the HTML document -->

<!ENTITY % list "UL | OL |  DIR | MENU">

<!ENTITY % preformatted "PRE">

<!ENTITY % block
     "P | %list | %preformatted | DL | DIV | CENTER |
      BLOCKQUOTE | FORM | ISINDEX | HR | TABLE | %shoe.content;">


<!-- Declarations for ontologies -->
<!ELEMENT ONTOLOGY      - -  (USE-ONTOLOGY |DEF-CATEGORY |
                              DEF-RELATION | DEF-RENAME |
                              DEF-INFERENCE | DEF-CONSTANT |
                              DEF-TYPE)* >

<!ATTLIST ONTOLOGY
        id              CDATA   #REQUIRED
        version         CDATA   #REQUIRED
        description     CDATA   #IMPLIED
        declarators     CDATA   #IMPLIED
        backward-compatible-with        CDATA   #IMPLIED >

<!ELEMENT USE-ONTOLOGY          - O  EMPTY >
<!ATTLIST USE-ONTOLOGY
        id              CDATA   #REQUIRED
        version         CDATA   #REQUIRED
        prefix          CDATA   #REQUIRED
        url             CDATA   #IMPLIED >

<!ELEMENT DEF-CATEGORY          - O  EMPTY >
<!ATTLIST DEF-CATEGORY
        name            CDATA   #REQUIRED
        isa             CDATA   #IMPLIED
        description     CDATA   #IMPLIED
        short           CDATA   #IMPLIED >

<!ELEMENT DEF-RELATION          - -  (DEF-ARG)* >
<!ATTLIST DEF-RELATION
        name            CDATA   #REQUIRED
        short           CDATA   #IMPLIED
        description     CDATA   #IMPLIED >

<!ELEMENT DEF-ARG               - O  EMPTY >
<!ATTLIST DEF-ARG
        pos             CDATA   #REQUIRED
        type            CDATA   #REQUIRED
        short           CDATA   #IMPLIED >
<!-- pos must be either an integer, or one of the strings:
     FROM or TO -->

<!ELEMENT DEF-RENAME            - O  EMPTY >
<!ATTLIST DEF-RENAME
        from            CDATA   #REQUIRED
        to              CDATA   #REQUIRED >

<!ELEMENT DEF-CONSTANT          - O  EMPTY >
<!ATTLIST DEF-CONSTANT
        name            CDATA   #REQUIRED
        category        CDATA   #IMPLIED >
```

```
<!ELEMENT DEF-TYPE              - O  EMPTY >
<!ATTLIST DEF-TYPE
        name            CDATA   #REQUIRED
        description     CDATA   #IMPLIED
        short           CDATA   #IMPLIED >

<!-- Delcarations for inferences -->
<!-- Inferences consist of if and then parts, each of which
     can contain multiple relation and category clauses -->
<!ELEMENT DEF-INFERENCE - - (INF-IF, INF-THEN) >
<!ATTLIST DEF-INFERENCE
        description     CDATA   #IMPLIED >
<!ELEMENT INF-IF                - -  (CATEGORY | RELATION |
                                   COMPARISON)+ >
<!ELEMENT INF-THEN              - -  (CATEGORY | RELATION)+ >
<!ELEMENT COMPARISON            - -  (ARG, ARG) >
<!ATTLIST COMPARISON
        op      (equal | notEqual | greaterThan |
                 greaterThanOrEqual | lessThanOrEqual |
                 lessThan)      #REQUIRED >

<!-- Declarations for instances -->
<!ELEMENT INSTANCE              - -  (USE-ONTOLOGY | CATEGORY |
                                   RELATION | INSTANCE)* >
<!ATTLIST INSTANCE
        key             CDATA   #REQUIRED
        delegate-to     CDATA   #IMPLIED >

<!ELEMENT CATEGORY              - O  EMPTY >
<!ATTLIST CATEGORY
        name            CDATA   #REQUIRED
        for             CDATA   #IMPLIED
        usage           (VAR | CONST)   CONST >
<!-- If VAR is specified for a category that is not within a
     <DEF-INFERENCE>, then it is ignored -->

<!ELEMENT RELATION              - -  (ARG)* >
<!ATTLIST RELATION
        name            CDATA   #REQUIRED >
<!ELEMENT ARG                   - O  EMPTY >
<!ATTLIST ARG
        pos             CDATA   #REQUIRED
        value           CDATA   #REQUIRED
        usage           (VAR | CONST)   CONST >

<!-- pos must be either an integer, or one of the strings:
     FROM or TO.  -->
<!-- If VAR is specified for an arg that is not within a
     <DEF-INFERENCE>, then it is ignored -->

<!-- Include DTD for HTML -->
<!ENTITY % HTMLDTD PUBLIC "-//W3C//DTD HTML 3.2 Final//EN" >
%HTMLDTD;
```

Appendix B: SHOE and XML

The Extensible Markup Language(XML) was designed to be a simplified version of SGML for the Internet, and is growing in popularity. SHOE was originally designed to be embedded in HTML pages, but since it uses an SGML syntax it can be used within XML documents with only a minimum of modifications. The XML version of the SHOE DTD can be found at http://www.cs.umd.edu/projects/plus/SHOE/shoe_xml.dtd. Before you can include the XML

version of SHOE in a document, you must make sure that the document is XML compliant. If it is an HTML document, then you make the document compliant with the W3C's recommendation on XHTML 1.0: The Extensible HyperText Markup Language. Then to include SHOE in the XHTML markup, simply follow the guidelines set forth in the W3C's Namespaces in XML Recommendation. Essentially, this means insert the following at the desired location in the document:

```
<shoe xmlns="http://www.cs.umd.edu/projects/plus/SHOE/" version="1.0">
```

and include the desired ONTOLOGY or INSTANCE tags between this tag and a `</shoe>` tag.

Due to the differences in between SGML and XML, SHOE XML is more restrictive and requires a slight variant in syntax than that described in the main section of this document.

- All *empty elements*, i.e., elements which have no content and no end tag, must end with a '/>' instead of a '>'. Specifically, this applies to the USE-ONTOLOGY, DEF-CATEGORY, DEF-ARG, DEF-RENAME, DEF-CONSTANT, DEF-TYPE, CATEGORY, and ARG elements.
- No attribute minimization is allowed. Therefore, when specifying VAR or CONST within the categories or arguments in an inference rule, the attribute name USAGE must be explicitly provided, e.g. `USAGE="VAR"` instead of `VAR`.
- Since XML is case-sensitive, all element and attribute names must be in lower case.
- All attribute values must always be quoted, including those which are numeric as well as the "FROM" and "TO" keywords.

SHOE can also be used independently of XHTML. A non-embedded SHOE XML document must begin with the appropriate XML prolog:

```
<?xml version="1.0"?>
<!DOCTYPE shoe SYSTEM
   "http://www.cs.umd.edu/projects/plus/SHOE/shoe_xml.dtd">
```

Web Accessibility