

Superimposed Schematics: Introducing E-R Structure for *In-Situ* Information Selections^{*}

Shawn Bowers, Lois Delcambre, and David Maier

OGI School of Science and Engineering at OHSU
Beaverton OR 97006, USA
{shawn,lmd,maier}@cse.ogi.edu

Abstract. Considerable research exists on providing structured access to unstructured information sources, primarily for search and query. Little attention has been placed on (1) keeping the by-products of the process (e.g., connections between structured and unstructured data are typically forgotten and context is lost), (2) developing a rich, conceptual structure for this new layer of information (e.g., the structured data is often represented in simple relational tables), and (3) further elaborating and linking the new, structured information. We propose superimposed schematics to address these issues. Superimposed schematics offer E-R modeling constructs integrated with marks, where a mark holds an address to an information element in an underlying source. A superimposed schematic enables a conceptual addressing scheme for the information it contains (directly) and for the information it references (through marks). This addressing scheme can then be used for marking from additional layers of superimposed information. We consider schematics as a useful model for superimposed information and discuss how it fits into our general research on superimposed information.

1 Introduction

The actions of the USDA Forest Service, such as selling timber or issuing or denying special use permits, are officially documented in *Decision Notices, Records of Decision*, and so forth. Members of the public then have the right to file an appeal requesting that the decision be changed. When the appeal deadline has passed, a *reviewing officer* from the Forest Service normally considers the entire set of appeals for a given decision and makes a recommendation. Finally, the *deciding officer* makes a determination for each issue raised (in one or more appeals). Each *appeal decision*¹ is typically represented in two standard letters, one from the reviewing officer and one from the deciding officer (see Figure 1 for an example decision memo).

^{*} This work supported in part by the National Science Foundation through grants EIA-99-83518 and IIS-98-17492. Information and examples concerning the Forest Service and appeal decisions provided by Timothy Tolle of the USDA Forest Service

¹ The Pacific Northwest Region of the USDA Forest Service maintains appeal decisions for twenty national forests online at <http://www.fs.fed.us/r6/plan/appeal.htm>, dating back to 1997.

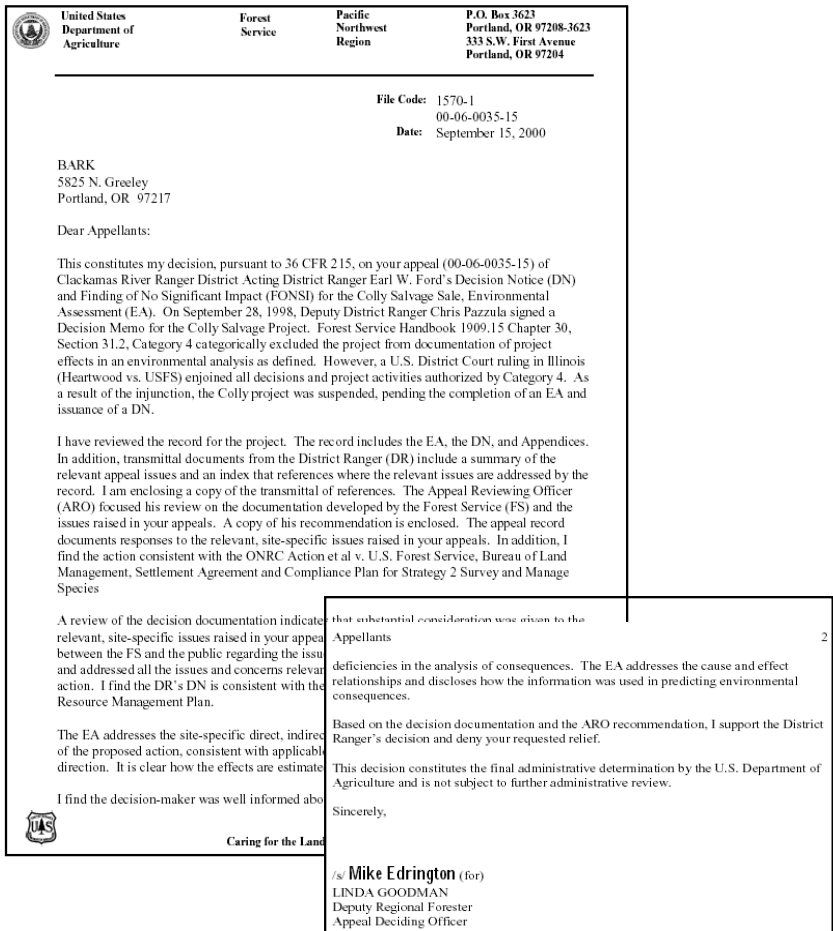


Fig. 1. The decision letter for an example appeal decision

Although not easy to discern from looking at the documents, an appeal decision is comprised of a fairly standard set of information items, such as: (1) which decision is being challenged, (2) which appellants have filed (and on behalf of which organization), (3) what issues were raised (across the set of appeals), and (4) what final determination was made for each issue. We observe that the standard items and relationships among them can be usefully represented in a *superimposed schematic*, an ER-style schema for superimposed information, as shown in Figure 2.

The key feature of a superimposed schematic is that it can be populated with *marks*, which are integral to superimposed information [4,5,6], in addition to regular attribute values. Each mark represents and holds the address for an information excerpt from the underlying information source (e.g., in a decision

letter), as shown in Figure 3. We see, for example, that a particular appeal is mentioned in the first paragraph of the decision letter and that the issue raised in that appeal is described by the entire third paragraph of the review document.

Appeal decisions are well suited for superimposed schematics because they are unstructured, heterogeneous sources of information with important, conceptual content (e.g., decision and review letters have little physical structure other than a typical memo). Schematics provide structured access at “no additional cost” [14], i.e., the underlying information does not require modification since schematics do not add semantic markup directly to the document. Schematics also offer more than simple nested hierarchies of document structure; they model exactly the concepts of interest (regardless of the documents structure) through an E-R schema. In general, any given document may have many associated schematics.

We consider document authors, schematic designers, schematic populators, and users of populated schematics as (potentially) separate people. For example, a schematic populator may create a new schematic instance for each appeal decision allowing natural resource managers (the end-users) to easily browse appeal decisions. A resource manager might begin with an Appeal Decision entity, navigate to the determination(s) and their associated issues, and then browse to see if the issue was raised on behalf of an organization.

Besides navigation, schematics enable a powerful mechanism for queries, which can be answered across a collection of schematic instances. When a natural resource manager is pondering a decision, she might like to know what issues were raised for similar decisions. And at a more strategic level, the USDA Forest Service routinely analyzes appeals to track the most important issues and trends. Both tasks are tedious and labor-intensive, requiring appeal decisions to be read and searched manually. Superimposed schematics are designed for both purposes: information browsing (introducing structure of interest over an unstructured universe of information) and collection-based querying.

Finally, most documents and their associated tools (like editors or browsers) provide limited addressing capabilities (e.g., page and byte offset, or section, paragraph, sentence hierarchies). One of our goals is to use superimposed schematics for *enhanced addressing* over underlying documents, i.e., to create addresses based on the conceptual information (in the schematic) for accessing document content. Enhanced addressing can be used for marking information elements in one schematic from additional layers of superimposed information. A resource manager can develop a schematic (as a new layer of superimposed information) for tracking important actions, issues, appeals, and results; and populate instances by using enhanced addresses to environmental assessment, scientific study, report, and appeal decision schematics.

In this paper, we present the data model for superimposed schematics and describe how schematics provide enhanced addressing for further levels of superimposed information. Our next step, which we are still currently developing, is to exploit schematics for query. The rest of this paper is organized as follows. In Section 2, we describe the data model for superimposed schematics. Section

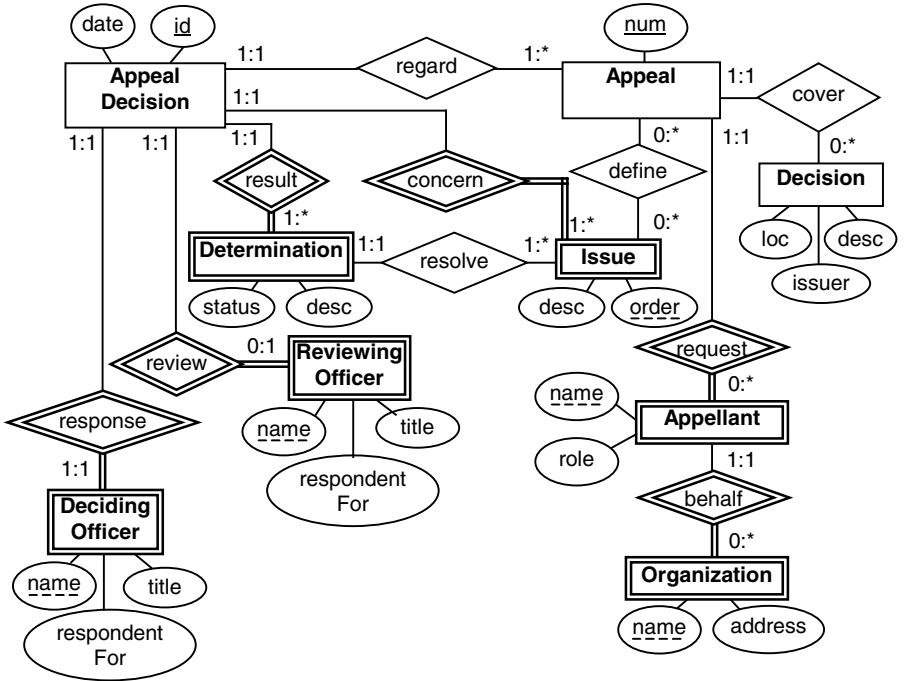


Fig. 2. A superimposed schematic representing the conceptual structure of the standard information elements in an appeal decision

3 presents an addressing language for superimposed schematics for supporting enhanced addressing. We discuss related work in Section 4, and conclude with a summary and a discussion of future work in Section 5.

2 A Data Model for Superimposed Schematics

The schematics data model is fundamentally based on Chen’s original E-R data model [3], however, to accomodate schematics, we permit multiple instances of a schema and most importantly, incorporate marks.

2.1 The Basic Schematics Data Model

Within a given domain, there may be multiple schematics of interest, which we maintain in a collection called the *schematic universe*. We define a schematic universe \mathcal{U} as a logical collection of schematics represented as the triple $(\mathcal{S}, \mathcal{ET}, \mathcal{RT})$, consisting of a set of named superimposed schematics \mathcal{S} , a set of named schematic entity types \mathcal{ET} , and a set of named schematic relationship types \mathcal{RT} . Each schematic $s \in \mathcal{S}$ consists of a set of schematic entity types in \mathcal{ET} such that \mathcal{S}

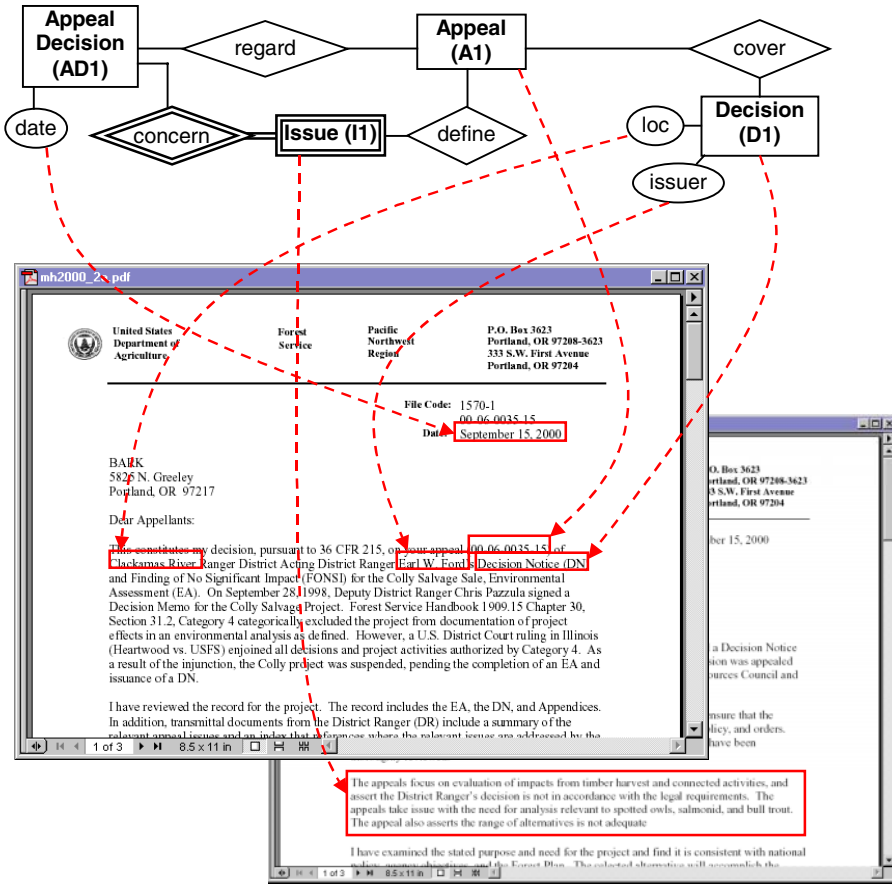


Fig. 3. A portion of decision memo (on the left) and accompanying review memo (on the right) shown with a portion of the populated superimposed schematic (at the top)

partitions \mathcal{ET} . Therefore, every schematic entity type in \mathcal{U} belongs to a superimposed schematic, and no two superimposed schematics contain the same entity type. Schematic relationship types are considered part of a schematic when all entity types they associate are within the schematic. Relationship types are not required to be within a schematic, i.e., we permit schematic relationship types that span schematics.

A schematic entity type $et \in \mathcal{ET}$ consists of a set of named attributes A . Each attribute in A has an associated domain represented by the function $dom : \mathcal{ET} \times \mathcal{A} \rightarrow \mathcal{VT}$, where \mathcal{A} and \mathcal{VT} represent the set of attribute names and value types in \mathcal{U} , respectively (note that if the attribute name is not defined for the entity type, dom returns *null*). We consider only single-valued attributes that have either atomic (e.g., strings and integers), complex (structured values

such as dates and phone numbers), or mark values. Additionally, we introduce the *uninterpreted type* as a valid value type, which says that the attribute can contain a value of any type including an unknown type (which is useful for certain kinds of marks).

A schematic relationship type $rt \in \mathcal{RT}$ is a multi-directional association between one or more entity types and is represented as a set \mathcal{L} of *roles*, each of which is a pair (rn, et) where rn is a role name and $et \in \mathcal{ET}$ an entity type. Roles represent the entity types associated by relationship types. For non-recursive binary relationship types, role names are optional (a missing role name is represented by *null*). However, all recursive and n -ary (where $n > 2$) relationships require role names. We do not permit attributes on relationships.

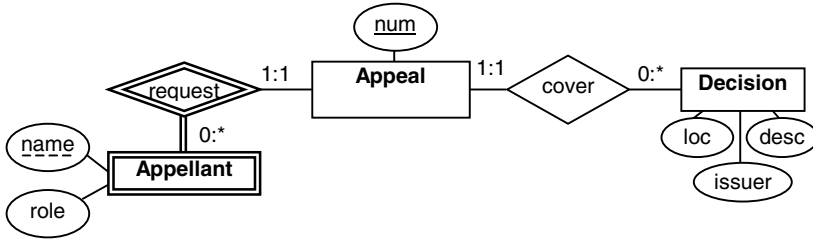
Instead of using the typical method of grouping entities into entity sets and relationships into relationship sets, we introduce an explicit extent for the schematic universe. The *universal schematic extent* \mathcal{D} is a triple $(\mathcal{I}, \mathcal{E}, \mathcal{R})$ that contains a set of schematic instances \mathcal{I} , a set of schematic entity identifiers \mathcal{E} (representing entities), and a set of schematic relationship identifiers \mathcal{R} (representing relationships). Each schematic instance $i \in \mathcal{I}$ contains a set of entities in \mathcal{E} such that \mathcal{I} partitions \mathcal{E} .

A schematic entity identifier e is a pair (et, V) , where $et \in \mathcal{ET}$ is the type of the entity and V a set of attribute-value pairs (att, v) . (Note the definition of type is more restrictive than permitted in most E-R models, where an entity can belong to more than one entity set, i.e., type.) For $V = \{(att_1, v_1), (att_2, v_2), \dots, (att_n, v_n)\}$, we require each att_i for $1 \leq i \leq n$ be unique such that if A is the set of attributes for et , $att_i \in A$ and $n \leq |A|$. That is, each attribute in V is unique and there is one attribute for each attribute of et .

A schematic relationship identifier r is a pair (rt, E) such that $rt \in \mathcal{RT}$ is the type of the relationship and E is a set of role name, schematic entity identifier pairs (rn, e) for $e \in \mathcal{E}$. Like entity attributes, each role name corresponds to a role name in rt 's set of roles \mathcal{L} . We limit relationships such that only one relationship of a given type can occur between a set of entity identifiers.

A schematic entity and a schematic relationship may additionally contain at most one *anchor* that holds a single mark (e.g., see Figure 3). Anchors are representing with the function $anchor : \mathcal{E} \cup \mathcal{R} \rightarrow \mathcal{M} \cup \{null\}$, where \mathcal{M} is the set of possible marks. The *anchor* function returns *null* if the identifier does not have an anchor. Anchors serve to “locate” an entity in an underlying source, when such a location is appropriate.

Attributes of an entity type can be specified as either *required* or *optional*. A required attribute implies that corresponding entities have non-null values for the attribute; optional attributes permit null values. Relationship types can specify minimum (required or optional) and maximum (*one* or *many*) cardinalities for each of their roles. Figure 4 defines a portion of the appeal decision schematic of Figure 2.



$\mathcal{U} = (\{AppealDecision\}, \{Appeal, Appellant, Decision\}, \{request, cover\})$.
 $AppealDecision = \{Appeal, Appellant, Decision\}$.
 $Appeal = \{num\}$ [key = {num}].
 $Appellant = \{name, role\}$ [partialkey = ({name}, request)].
 $Decision = \{loc, issuer, desc\}$.
 $request = \{(null, Appeal), (null, Appellant)\}$.
 $cover = \{(null, Appeal), (null, Decision)\}$.
 $dom(Appeal, num) = String$.
 $dom(Appellant, name) = String$.
 $dom(Appellant, role) = String$.
 $dom(Decision, loc) = String$.
 $dom(Decision, issuer) = String$.
 $dom(Decision, desc) = String$.

Fig. 4. Part of the appeal decision schematic described using the model. Note that only keys and partial keys are assumed to be required

2.2 Marks and Constraints

A key feature of superimposed schematics is that it integrates marks into an E-R style model. We examine here the effect of marks on type conformance checking and on the definition of keys.

To check whether an entity conforms to its corresponding entity type, we must take into account marks, since attributes can have marks as values. We require that all marks support the function *excerpt* : $\mathcal{M} \rightarrow \mathcal{V}$, where \mathcal{V} is the set of all possible values. The default type of a mark’s excerpt is the uninterpreted type. If an attribute has a mark as a value and the attribute’s domain is not the uninterpreted type, we must *interpret* the type of the excerpt to check conformance. Note that this interpretation is done when the mark is assigned as the attribute value. There are a number of ways in which an excerpt can be interpreted, for example:

1. The base layer may support a function for supplying the excerpt’s type (e.g., whether it is a text string, image, or some other type).
2. The base layer may be able to provide a value with a known type to represent the excerpt (e.g., a text string representing the text in an image).
3. An external or user-defined function could be used (e.g., to convert the excerpt into a structured value).
4. The creator of the mark could provide a typed value to represent the excerpt.

In most cases, the type of an excerpt will be a simple value like a string, which is the case for appeal decisions.

Once an excerpt’s type has been interpreted, we test conformance in the normal way. That is, we ensure all required attributes are present and that each attribute value of the entity conforms to the domain of the corresponding attribute name of the entity type. Relationship conformance is also checked in the typical way, i.e., each entity associated by a relationship is required to have the same type as the corresponding role of the relationship type.

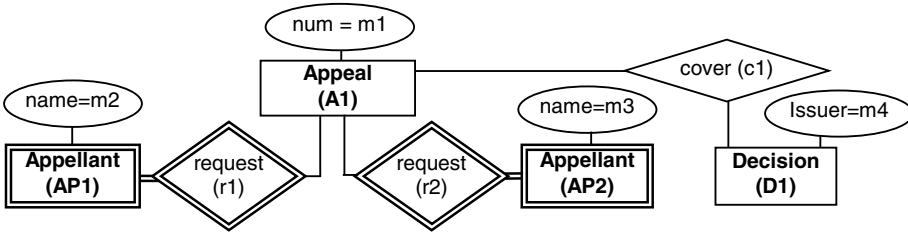
In the schematics model, key constraints on entity types are permitted as are weak entity types with (or without) partial keys. A key constraint is denoted by a set $K \subseteq A$, where A is the set of attribute names for the entity type. An entity type is not required to have a key constraint. Without a key, two entities with different entity identifiers and identical values can exist in the universal extent. A partial key is defined as a pair (K, rt) , where K is the partial key and rt is a binary, identifying relationship type. A key definition constrains entities in the normal way, i.e., only one entity of the entity type in the universal extent \mathcal{D} can contain the key value. However, like with entity type conformance, we must take into account marks that are defined to be part of the key. Testing equality between entities with mark-valued attributes in the key requires determining if two marks (as values for an attribute of an entity) are equal. Normally, the base layer will provide a Boolean-valued function for testing equivalence of marks. However, if no such function exists, (as in conformance) we treat a mark’s value as its excerpt’s value. This approach considers two entities with identical key values, where at least one of the key values is an excerpt, but with different marks, as identical.

Keys in superimposed schematics have a subtle consequence when mixed with marks. For example, consider the appeal decision schematic of Figure 2. The Reviewing Officer entity type would normally have a key (perhaps its name attribute). However, we want to store exactly one mark (for the officer’s name) for each appeal decision that the officer reviews. Further, each such “occurrence” of the officer’s name could be spelled slightly differently (e.g., sometimes with a middle initial). We introduce *authoritative entity types* to represent key values in these situations. That is, an authoritative entity type serves as the domain for the key of one or more other entity types. Relationships connect entities to their corresponding authorities (via *authoritative relationship types*) to denote identity and enable queries with unique values.

As an example, we could define a new schematic to represent directory information that contains an authoritative entity type called *Person* (with a keyed attribute called “name”). In this way, we can add a relationship type from the Reviewing Officer entity type to the Person authoritative entity type to relate Officers to their corresponding Person. Thus, we can find all decisions where a particular person served as a Reviewing Officer.

2.3 Schematic Instances

A *schematic instance* consists of the entities and associated relationships that together represent one populated context for a schematic. For example, an appeal decision schematic instance consists of one Appeal Decision entity, its Reviewing



$D = (\{I_1\}, \{A_1, AP_1, AP_2, D_1\}, \{r_1, r_2, c_1\})$.
 $I_1 = \{A_1, AP_1, AP_2, D_1\}$.
 $A_1 = (Appeal, \{(num, m_1)\})$.
 $AP_1 = (Appellant, \{(name, m_2)\})$.
 $AP_2 = (Appellant, \{(name, m_3)\})$.
 $D_1 = (Decision, \{(issuer, m_4)\})$.
 $r_1 = (request, \{(null, A_1), (null, AP_1)\})$.
 $r_2 = (request, \{(null, A_1), (null, AP_2)\})$.
 $c_1 = (cover, \{(null, A_1), (null, D_1)\})$.
 $excerpt(m_1) = "00-06-0035-15"$.
 $excerpt(m_2) = "John Rancher"$.
 $excerpt(m_3) = "Gregory J. Dyson"$.
 $excerpt(m_4) = "Earl W. Ford"$.

Fig. 5. Part of a populated appeal decision schematic. Note that each entity must be associated with an entry point (i.e., an Appeal Decision entity for this example)

and Deciding Officers, and a set of Appeals, each with a Decision, Appellants, and Issues with Determinations. A schematic can have zero or more instances. Part of a schematic instance for appeal decisions is shown in Figure 5.

To identify schematic instances, we introduce the notion of an *entry point*. Any entity type with a key can serve as an entry point. For entity type ET designated as an entry point in a schematic s , each schematic instance of s must contain *one and only one* entity of type ET . Entry points serve as schematic-instance keys—they allow schematic instances to be uniquely identified. For example, the Appeal Decision entity type serves as an entry point, thus allowing the *id* attribute (of the Appeal Decision entity type) to uniquely determine an instance of the appeal decision schematic.

We require every schematic to define at least one entry-point entity type. Whenever an entity whose type is an entry point is created, a new schematic instance is defined. Alternatively, each newly created entity whose type is not considered an entry point must be explicitly placed within a schematic instance. For example, when a new Determination entity is created, an existing Appeal Decision entity is given (using an *id* attribute value), and the Determination is then placed in the corresponding schematic instance. Note that we do not require every entity in a schematic instance to be related to another entity. Finally, we define the function $instance : \mathcal{E} \cup \mathcal{R} \rightarrow \mathcal{I}$, which takes an entity or relationship identifier and returns the identifier’s corresponding schematic instance.

3 Schematic-Based Addressing

To address selections in underlying documents, superimposed schematics use marks. However, there are also cases where marking into a schematic is desired. Consider Figure 6, which shows a virtual document of selected issues, excerpted from a set of appeal decisions. Each excerpt is displayed along with a mark to the corresponding issue to enable user navigation into the associated schematic instance (which can then be further navigated, e.g., to find determinations or to see who the appellants are). In general, marks into schematics permit additional layers of superimposed information. This section describes our approach for supporting marks into superimposed schematics.

3.1 Opaque versus Transparent Marks

A mark is an address to an information selection made within a particular context. We require all marks to support the operation *resolve*, which dereferences a mark by opening and (possibly) highlighting the corresponding information selection. For example, for a particular issue in an appeal decision, we can call *resolve* on its anchor to view the appropriate sentence(s).

Just as an underlying document is considered a context, we also view a schematic instance as a context. For example, a schematic instance for an appeal decision groups exactly the information necessary to represent a single appeal decision (even though there may be many underlying documents). For marks into schematics, *resolve* should open the associated schematic instance and highlight the selection (i.e., entity, relationship, or attribute). In cases where multiple levels of superimposed information exist, we may also wish to have finer control over *resolve*, e.g., to resolve a mark into a schematic by unnesting the mark until we reach an anchor into a base layer document. We provide the *unnest-mark* : $\mathcal{M} \rightarrow \mathcal{M}$ function, which takes a mark and returns the mark it refers to (e.g., the entity or relationship anchor) or *null* if no such mark exists.

We consider two types of marks, *opaque* and *transparent*. An opaque mark contains an address that is application specific. For example, an address generated by MS Excel or MS Powerpoint is an opaque mark. By transparent mark, we mean marks whose addresses are semantically meaningful. Common examples of transparent marks include URLs and XPath/XPointer addresses. An advantage of transparent marks is that they can be created and usefully examined out of context. Also, opaque marks can only be created within the associated base-layer application, whereas transparent marks can be created without any application intervention, i.e., they can be created in the superimposed layer.

Our goal is to define an addressing scheme over schematics to support transparent marks. We take a conservative approach, namely, we restrict an address to refer to a single entity, relationship, or attribute. We impose this constraint since the model for schematics does not support arbitrary collections, e.g., we support single valued attributes (not multi-valued) and implied collections exist only as schematic instances and the universal schematic extent.

3.2 Addressability Constraints

Depending on the structure of a schematic, not all entities, relationships, and attributes can always be uniquely addressed. To determine which items can be transparently addressed, we introduce the *schematic addressability constraint*. Intuitively, to address an entity it must be identifiable. An entity cannot be uniquely identified if it does not have some form of key or is not reachable via a unique path from a keyed entity. This situation also applies to relationships, since a relationship is identified by the entities it associates. Only *identified* entity types are transparently addressable. We recursively define an identified entity type as follows:

1. An entity type is an identified entity type if it has a key (which includes entry points).
2. A weak entity type is an identified entity type if it has a partial key.
3. An entity type $e1$ is an identified entity type if it uniquely participates with an identified entity type $e2$. (By uniquely participates, we mean that there is a binary relationship type that associates $e1$ through role $r1$ to $e2$ through role $r2$ such that $r1$ has a maximum cardinality of one.)

To be a *completely addressable* schematic, every entity, relationship, and attribute must be transparently addressable. To be completely addressable, we must guarantee that all entity types be identified (via the addressability constraint). Second, we must guarantee that all entities with entity types that are identified only through a unique participation relationship type (part 3 above) are associated by such a relationship. In this way, we can uniquely address every entity through either its key, its identifying entity's key and its partial key, or through its unique participation entity's key (since the relationship is guaranteed to exist).

We do not require schematics to be completely addressable nor do we require that the addressability constraint be met. For some applications, enhanced addressing will not be a priority, e.g., applications focused on navigation and browsing. Opaque addresses, however, can always be defined for entities and relationships by using their identifiers. Therefore, all entities, relationships, and attributes can be (at least) opaquely addressed. Note that the appeal decision schematic in Figure 2 does not satisfy the addressability constraint since the *Decision* entity type is not an identified entity type. However, if *Decision* were an identified entity type, the schematic would satisfy the addressability constraint.

Finally, all marks are required to support the *excerpt* function, therefore, we must consider excerpts for marked entities, relationships, and attributes. For an attribute, we can use its value, or if it has a mark, its mark's excerpt. For an entity or relationship, the anchor's excerpt can be used (if such an anchor exists).

3.3 The Schematic Addressing Language

A transparent schematic address P is a list of address components p_i for $1 \leq i \leq n$ such that P has the form $p_1.p_2. \dots .p_n$ (see Table 1 for examples). We

distinguish between three kinds of schematic addresses, namely, entity addresses, attribute addresses, and relationship addresses, each of which are defined below.

Entity Address A valid entity address is defined recursively as:

1. *Key Entity*. The address $et(\kappa)$ is a valid address if et is an entity type with key K and κ is a key value of the form $k_1 = v_1, k_2 = v_2, \dots, k_m = v_m$ for all $k_i \in K$ and $i \geq 1$ and $m = |K|$. The type of the entity addressed is et .
2. *Partial Key Entity*. The address $P.rt(\kappa)$ is a valid address if P is a valid entity address for an entity of type $et1$, rt is an identifying relationship between $et1$ and $et2$, and κ is a partial key value for $et2$. The type of the entity addressed is $et2$.
3. *Unique Participation Entity*. The address $P.rt$ is a valid address if P is a valid entity address for an entity of type $et1$ and rt is a unique participation relationship from $et1$ to $et2$. The type of the entity addressed is $et2$.

Relationship Address A valid relationship address is defined recursively as:

1. *Single Participation Address*. The address $P.rt\#$ is a valid address if P is a valid entity address of type et and rt is a binary relationship type connecting et to et' such that the role et' plays has a maximum cardinality of one.
2. *Multiple Participation Address*. The address $P.rt\#[P']$ is a valid address if P is a valid entity address of type et , rt is a binary relationship type between et and et' such that the role played by et' in rt has a cardinality of *many*, and P' is a valid entity address of type et' . Intuitively, to identify the *many* end of a many-to-many or one-to-many relationship type rt , we must identify the entity that the corresponding relationship connects.
3. *Non-Binary Address*. The address $P.rt\#/role_1[P'_1]/\dots/role_m[P'_m]$ is a valid address if P is a valid entity address of type et , rt is an n -ary (for $n > 2$) relationship type from et to a list of entity types $[et_1, et_2, \dots, et_m]$, $role_i$ for $1 \leq i \leq m$ is the role played by et_i , and P'_i is an entity address of type et_i .

Attribute Address The address $P.a$ is a valid address if P is a valid entity address of type et and a is a valid attribute name for et .

To determine the schematic instance of an entity or relationship address, e.g., for implementing the *resolve* operation, we can use the *instance* function defined in the previous section. For attribute addresses $P.a$, we can call *instance* on P to determine the schematic instance. Examples of transparent addresses over the Appeal Decision schematic are given in Table 1.

Finally, Figure 6 shows an example virtual document serving as a new layer of superimposed information over the Appeal Decision schematic. The virtual document uses enhanced addressing—it stores excerpts of issues from schematic instances as well as transparent marks to Issue entities. The virtual document demonstrates a notional example of query, i.e., the virtual document results from

Table 1. Example enhanced addresses. Note constants x and y represent “1570-1-0032-10” and “00-06-0032-10,” respectively, and AD stands for `AppealDecision`

<i>Address</i>	<i>Description</i>
<code>AD(id=x).concern(order=1)</code>	Address to an <i>Issue</i> entity.
<code>AD(id=x).concern(order=1).desc</code>	Address to an Issue <i>desc</i> attribute.
<code>AD(id=x).response#</code>	Address to a <i>response</i> relationship.
<code>AD(id=x).result#[Appeal(num=y).define(order=1).resolve]</code>	Address to a <i>result</i> relationship (identified via its associated <i>Issue</i> , as shown in brackets).

<p>Issue [<code>AppealDecision(id="1570-1-0032-10").concern(order=1)</code>] The decision violates the Federal Advisory Committee Act (FACA). The illegal advisory committee played a central role in influencing the Forest Service Decision.</p> <p>Issue [<code>AppealDecision(id="1570-1-0032-10").concern(order=2)</code>] The decision significantly reduces hunting opportunity and habitat for deer and elk.</p> <p>Issue [<code>AppealDecision(id="1570-1-0035-13").concern(order=1)</code>] The District Ranger’s decision is not in accordance with the legal requirements of the National Environmental Policy Act, the National Forest Management Act, and the Forest and Rangeland Renewable Resources Planning Act.</p>
--

Fig. 6. A virtual document with enhanced addresses to corresponding Issues

the schematic query: “List all Issues concerning Decisions made in the Mount Hood National Forest within the last two years.”

4 Related Work

Superimposed models differ in how they structure information. A number of untyped, hypertext models (such as HTML) are unstructured. The bundle/scrap model [6] along with most annotation models [14,16] provide very simple structures for organizing superimposed data. For example, in the bundle/scrap model, marks are held in named “scraps” and can be placed into (possibly) nested, named bundles. Semi-structured models are the most prevalent superimposed models, and include XLink [7] (for XML), RDF [10], Topic Maps [2], and hypertext models for emergent structure [11]. Superimposed schematics are richly structured, and to our knowledge the only approach based on the E-R model, which offers a standard, powerful conceptual data model. Other structured models include Structured Maps [4] (a simple, typed version of a Topic Map), typed hypertext systems [13], and models for wrappers [1,9,15].

Superimposed models also differ in their support for marks. Opaque marks are derived from the mark architecture in the bundle/scrap model [6], and support fine-grain, sub-document marks to arbitrary sources. Multivalent Documents [14] provide a document model (that various document formats are mapped to) for robust marks, but marks must be resolved using a special viewer.

Topic Maps, RDF, and HTML support marks through URLs, which are limited for sub-document addressing (using HTML document fragments). Although XPath/XPointer addresses can be used, they only support marks into XML documents. Schematics support transparent marks, similar in spirit to the motivation for XPath/XPointer. However, XPath/XPointer addresses are based on document structure (since they address XML) as opposed to conceptual structure. Note that wrappers offer a limited form of superimposed information since they support new layers of information, but not marks.

The constructs used for linking (i.e., marking) also differentiate superimposed models. Unique to the bundle/scrap model and superimposed schematics is the notion of context to separate layers of information. That is, a mark is a separate modeling construct used to navigate into a different context (both conceptually and operationally). While a number of approaches [2,4,7,10,12,14,16] store superimposed information separately from the base layer, they usually operate as if both layers were within the same context. For example, Structured Maps, XLink, typed hypertext [13], and most Topic Map applications integrate superimposed information into a single hypertext (using HTML).

Finally, some superimposed models support the use of excerpts, e.g., with virtual documents [12]. We have also encountered examples of special purpose, hard-wired schematics, such as in the Distributed Annotation Server [8] in which sequence “landmarks” are used to attach annotations to genetic information.

5 Summary and Future Work

Superimposed schematics contribute a rich conceptual layer for representing and elaborating experted information from underlying documents, while maintaining context through marks. We have presented a data model for schematics based on E-R structures with support for marks, multiple schematic instances (via entry points), and an accompanying addressing language.

We have implemented a browser for superimposed schematics, which can be used to navigate schematic instances and underlying information. We intend to enhance the browser for mixing navigation and query. We believe simple techniques can be used to help populate schematics semi-automatically. In addition, we believe integrating schematics into a document editor to tie population with actual document creation would also be useful. Finally, we intend to further explore enhanced addressing. Currently, we are working towards a model for layers of schematics over appeal decision packets. Appeal packets are used by deciding officers and others to assess an entire project, its issues, alternatives, and so on. We also wish to explore using enhanced addressing for standard E-R schemas, possibly to support tracking data lineage.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proc. of the 19th VLDB Conf.*, pp. 73–84, Aug. 1993. [102](#)
2. M. Biezunski, M. Bryan, S. Newcomb, eds. ISO/IEC 13250, *Topic Maps*, 2000. <http://www.ornl.gov/sgml/sc34/document/0058.htm>. [102](#), [103](#)
3. P. P. Chen. The entity-relationship model—toward a unified view of data. In *ACM Trans. on Database Systems*, 1(1):9–36, 1976. [93](#)
4. L. Delcambre, et al. Structured maps: Modeling explicit semantics over a universe of information. In *Intl. Journal on Digital Libraries*, 1(1):20–35, 1997. [91](#), [102](#), [103](#)
5. L. Delcambre and D. Maier. Models for superimposed information. In *Advances in Conceptual Modeling: ER '99 Workshop on the WWW and Conceptual Modeling*, LNCS 1727, pp. 264–280, Nov. 1999. [91](#)
6. L. Delcambre, et al. Bundles in captivity: An application of superimposed information. In *Proc. of the IEEE Intl. Conf. on Data Eng. (ICDE)*, pp. 111–120, Apr. 2001. [91](#), [102](#)
7. S. DeRose, E. Maler, D. Orchard, B. Trafford, eds. XML Linking Language (XLink), W3C Working Draft 21-Feb-2000. <http://www.w3.org/TR/2000/WD-xlink-20000221>. [102](#), [103](#)
8. R. D. Dowell, et al. The distributed annotation system. In *BMC Bioinformatics*, 2:(7), Oct. 2001. <http://www.biomedcentral.com/1471-2105/2/7> [103](#)
9. D. W. Embley, et al. A conceptual-modeling approach to extracting data from the web. In *Proc. of the 17th Intl. Conf. on Conceptual Modeling*, LNCS 1507, pp. 78–91, Nov. 1998. [102](#)
10. O. Lassila and R. R. Swick (eds.). Resource Description Framework (RDF) Model and Syntax Specification, W3C Rec. 22-Feb-1999, <http://www.w3.org/TR/REC-rdf-syntax>. [102](#), [103](#)
11. C. C. Marshall, F. Shipman, and J. Coombs. VIKI: Spatial hypertext supporting emergent structure. In *Proc. of the European Conf. on Hypertext Technology (ECHT)*, pp. 13–24, Sept. 1994. [102](#)
12. S. H. Myaeng, et al. A digital library system for easy creation/manipulation of new documents from existing resources. In *Proc. of the 6th Conf. on Content-Based Multimedia and Information Access (RIAO)*, Apr. 2000. [103](#)
13. J. Nanard and M. Nanard. Should anchors be typed too? An experiment with MacWeb. In *Hypertext'93 Proceedings*, pp. 51–62, Nov. 1993. [102](#), [103](#)
14. T. Phelps and R. Wilensky. Multivalent documents. In *Communications of the ACM*, 43:(6):83–90, June 2000. [92](#), [102](#), [103](#)
15. A. Rajaraman and J. D. Ullman. Querying websites using compact skeletons. In *Proc. of the 20th ACM Symposium on Principles of Database Systems*, pp. 16–27, May 2001. [102](#)
16. M. Rosheisen, C. Mogensen, T. Winograd. Shared web annotations as a platform for third-party value-added information providers. STAN-CS-TR-97-1582, Stanford Integrated Digital Library Project, Nov. 1997. [102](#), [103](#)