2 A Declarative Approach to Entity Resolution

Tanton H. Gibbs

_Acxiom Corporation_
_Little Rock, AR, USA_

2.1 Introduction

As companies gather and process more data from disparate sources, they are relying more heavily on entity resolution. Currently, creating an entity resolution system is a very procedural process. Blocking, transitive closure, and matching must all be pieced together whether by an Extract, Transform, and Load (ETL) tool or by a custom program (Galhardas et al. 2000). This is similar to the state of data querying before the advent of the Structured Query Language (SQL). In this chapter, a declarative approach to entity resolution is presented that gives the user the ability to specify what he or she would like resolved while allowing a code generator to determine the best way to resolve it. This chapter does not explore algorithms for blocking, transitive closure, clustering, or matching, but instead refers to papers on those subjects written by other authors (Baxter et al. 2003; Gu and Baxter 2004; Winkler 2000, 2003; Jaro 1989; Bhattacharya and Getoor 2006). Instead a background and defense of entity resolution and declarative languages is presented with a declarative solution and a possible representation. In section 2.2, a background of entity resolution is given. This covers both the rationale for and components of an entity resolution system. Section 2.3 begins the explanation of the declarative approach, focusing on the nouns of the taxonomy presented in this chapter. Section 2.4 continues the explanation, focusing on the adjectives followed by section 2.5 describing the verbs. The nouns, adjectives, and verbs are combined into an example representation in section 2.6 and then the chapter concludes in section 2.7.
2.2 Background

2.2.1 Entity Resolution Definition

Entity resolution is a fairly young field that stemmed from the commercial need for Customer Data Integration and record linkage. Record linkage is the task of identifying identical records across one or more record sources (Fellegi and Sunter 1969). It is also known as merge/purge (Hernandez and Stolfo 1995). Traditionally, record linkage was based around the Fellegi-Sunter statistical approach (Fellegi and Sunter 1969). This approach has been re-fined over the years to include modifications to approximate string matching (Winkler 1990) as well as the ability to better predict weights using an Expectation Maximization algorithm (Winkler 1988).

In recent years, the focus of record linkage has broadened to include training data (Cohen and Richman 2002), clustering techniques (McCallum et al. 2000), and even vector spaces (Jin et al. 2003) in order to bypass some of the deficiencies of the Fellegi-Sunter statistical approach.

Entity resolution can be viewed as an extension to record linkage that views the problem as a database problem (Benjelloun et al. 2006a) and expands the problem to include multiple entities and their associations (Bhattacharya and Getoor 2006).

2.2.2 Entity Resolution Defense

Data integration is quickly becoming one of the top needs of both public and private companies. From federal security to marketing campaigns, understanding the customer’s interactions with your agency or company is vital. Moreover, both real-time and offline entity resolution are required for today’s businesses. Business call centers need to be able to access a customer’s accounts across lines of business in order to understand the caller’s relationship with the company. This interactive requirement necessitates a real-time component to the entity resolution framework. In addition, marketing campaigns may require the processing of millions or billions of prospect records. The computational requirements for this may necessitate different algorithms (Benjelloun et al. 2006b) and may need to be run offline so that interactive access can proceed with less latency. In other words, today’s entity resolution system is the heart of the enterprise, equipping analysts and support representatives with the knowledge they need to do their jobs. To function effectively, it must be high throughput, low latency, flexible, scalable, and cost efficient.
2.2.3 Entity Resolution Terminology

The rest of this section explores existing entity resolution terminology. Creating a declarative language requires keen attention to the meaning of the terms used in that language. In this section, we will examine some of the operations regarding an entity resolution implementation since we will not describe those operations later. If the reader is interested in the implementation, he or she may follow the references in this section.

2.2.3.1 Prospecting

If performance is not a concern, or if the number of records is small, then it is satisfactory to match every record against every other record. In fact, many solutions propose to do that or minimize the cost of that (Benjelloun et al. 2006a, 2006b; Fellegi and Sunter 1969). Unfortunately, in today’s world of massive datasets, it is often infeasible to perform that level of matching. Instead, entity resolution systems resort to prospecting. **Prospecting** is a function that quickly reduces the number of records that need to be sent to the matching engine. If we use the old analogy of finding a needle in a haystack, the goal of prospecting is to reach a hand in and pull out the needle with as little extra hay as possible.

2.2.3.2 Blocking

One common technique that is used for prospecting is known as blocking (Hernandez and Stolfo 1995; McCallum et al. 2000). **Blocking** segregates the input file into sections, or windows. Each record has its own window, or section of the file, that should contain the record of interest. Obviously, the wider the window is, the more sure you are that the record of interest is in the window. However, the wider the window is, the more records that you must match against and the longer it will take to get an answer.

2.2.3.3 Closure

Another common prospecting approach is closure. **Closure** treats the records as vertices in a graph with commonalities as the edges. It then uses the graph’s connected components as the prospect set. So, if record A shares a phone number with record B, which shares an address with record’s C and D, then all of those records would be connected and would be members of the prospect set. Another view of closure is shown by Bhattacharya and Getoor (2005). The authors view closure as a hyper-edge connecting entity references by associations.

As seen in Bhattacharya and Getoor (2006), closure can be computed on the fly to a specified depth. Limiting the depth helps prevent irrelevant or promiscuous data from expanding the prospect set. For instance, an apartment might have a large number of occupants across time, so its records derivatives should be quickly pruned from further analysis. It is worth noting that the results of the on-the-fly closure are stored so that future queries can be performed faster.
2.2.3.4 Matching

Matching is a function that determines whether two records represent the same real-world entity. Historically, approximate string matching (ASM) has been the center of the record linkage universe (Fellegi and Sunter 1969; Jaro 1989; Winkler 1990; Navarro 2001). With entity resolution, ASM is still important, but the system must also consider associations that ASM cannot handle such as changes of address, married name changes, aliases, and address standardization changes. In (Benjelloun et al. 2006a), matching is treated as a generic function that could encapsulate all of those associations. Furthermore, clustering approaches to entity resolution include matching, but also focus on the structural similarities between entity graphs (Bhattacharya and Getoor 2006).

2.2.4 Declarative Languages

A declarative language focuses on what the user wants accomplished rather than how the user wants it accomplished. A language such as the SQL represents many complicated procedural statements in just a few high level sentences. “SELECT * FROM TABLE WHERE X = Y” has behind it numerous optimizers, cache detection algorithms, file reading algorithms, etc…, but all of that is hidden from the SQL user.

A declarative language is, by nature, domain specific. The language uses business terminology to encapsulate the procedural statements into a form that is widely recognized and understood by the business analysts. For example, in Galhardas et al. (2000), the author uses a SQL-like language to provide a declarative language for the data cleansing domain, replacing a traditionally ETL-driven process with a concise, descriptive language.

2.3 The Declarative Taxonomy: The Nouns

The primary contribution of this chapter is the declarative taxonomy that begins in this section and continues through section 2.5. This taxonomy gives precisely defined terms that can be flexibly implemented to create a tailor made entity resolution system. This section focuses on the taxonomy’s nouns. Some of the nouns currently exist in entity resolution literature. Others were created and defined in conjunction with the taxonomy. This section will distinguish between the two where appropriate.

The taxonomy’s nouns promote the needs of businesses that offer many services around the same set of data. For instance, customers might be linked together based on individual attributes in order to determine credit worthiness. However, those same customers might be linked together into a household to ensure only one flyer gets sent to each family. The taxonomy clearly separates incoming records (what we will call references) and the entities that they belong to, thereby allowing a more flexible system. In the following subsections, we will
explore the nouns of the declarative taxonomy: *attributes, references, paths and match functions, entities, super groups, and matching graphs*.

2.3.1 Attributes

Attributes are the building blocks of the other nouns. They are atomic values that represent such constructs as *first-name, last-name, street, SSN*, or *date-of-birth*. An attribute may also identify a specific collection of attributes due to database normalization (e.g., a key representing all the attributes of an address [street, city, state, zip] that are located in a separate address table). An attribute may also be algorithmically generated. For example, search keys are often generated to facilitate fast lookup of records with similar name or address attributes. On its own, an attribute is without context, or additional information that provides meaning, and therefore, to be useful, attributes must be collected together. In other papers, attributes are known as fields (Singla and Domingos 2006), features (Benjelloun et al. 2006a), or attributes (Benjelloun et al. 2006a).

2.3.2 References

The taxonomy defines a reference as a heterogeneous, fixed length collection of attributes. A reference typically represents a real world person, place, or thing. More specifically, a reference represents someone’s representation of a real world person, place, or thing. For instance, a reference given by a web site might have only a username and email address. However, a reference to the same person given by a bank will likely have much more information, such as account number, social security number, and street address. Therefore, references are scraps of information, or clues, that must be analyzed to piece together the entity resolution puzzle. The taxonomy defines the term *input reference* to mean information that is input into the entity resolution system, whereas a *managed reference* is information that has been processed and persisted by the entity resolution system and will be managed over time. Each managed reference can be identified by a unique reference link. In other papers, the term record is used because the data is often a record in a database or file and because of the historical tie between entity resolution and record linkage (Benjelloun et al. 2006a; Winkler 1988). However, the taxonomy uses the term reference because the data is a reference to one or more entities, whether or not it exists as a unique record (Benjelloun et al. 2006a). A UML view of references and attributes can be seen in Figure 2.1.
Fig. 2.1. The relationship between references and attributes is many to many.

2.3.3 Paths and Match Functions

To find a managed reference when presented with an input reference, the entity resolution system follows one or more paths and uses a match function. A path is a collection of prospectors coupled with a well defined traversal sequence. In addition, a path has one or more required attributes that must exist on the input reference before the entity resolution system will follow the path. If the required attributes exist on the input reference, then that path is said to be valid, and the entity resolution system will use the path’s prospectors to search for similar records. Intra-path prospecting is short-circuited when prospects are found. Each prospector is tried until a prospector returns a set of managed references. The resulting (possibly empty) managed reference set is the output set of the path. If more than one path is valid, then each path’s output sets are unioned to form the final set of candidate references. The input reference and the candidate references are then fed into the match function (Benjelloun et al. 2006a, 2006b) to determine the most closely matching managed reference. The intra-path logic for two prospectors is shown by the UML sequence diagram in Figure 2.2. Inter-path logic is shown by the UML sequence diagram in Figure 2.3. It is worth noting that while there are multiple prospectors per path and multiple paths per reference type, there is only one match function per reference type. This is shown visually in the UML diagram in Figure 2.4.
Fig. 2.2. A UML sequence diagram representing the logic of a single path

1: Valid := isValid()
2: [Valid = true] : fetchProspects()
3: prospects = fetchProspects()
4: [prospects.size() = 0] : prospects = fetchProspects()
5: prospects

Main Path Prospector 1 Prospector 2

Fig. 2.3. A UML sequence diagram showing the logic when multiple paths are involved.

1: prospects1 := fetchProspects()
2: prospects2 := fetchProspects()
3: prospects := prospects1 ∪ prospects2
4: bestMatch := match(inputRecord, prospects)
2.3.4 Entities

The core of an entity resolution system is the entity (Benjelloun et al. 2006a, 2006b; Bhattacharya and Getoor 2005; Singla and Domingos 2006). An entity is a homogeneous, variable length collection of managed references. A good example of an entity is a consumer. Business-to-consumer companies need to keep track of their customers, or consumers, in order to effectively market to them. They need to understand the call center’s reference to “Bob Smith 123 Main St. 21543” represents the same consumer as the web site’s reference to “bsmith@isp.com.” To help them solve that critical problem, an entity resolution system assigns the same entity link to both references. To find an entity link for an input reference, the entity resolution system finds the best matching managed reference and uses its associated entity link. An entity match function is used to determine if two different managed references should have the same entity link. This match function accepts two managed entities and analyzes them to determine if they belong to the same entity. The entity match function will often be different from the reference match function. For example, for a household entity, Bob and Melinda Smith at the same address may represent a valid match. However, it is unlikely that these two would match with regards to the reference’s match function. This example shows how a reference may belong to two different entities (Consumer and Household). Nevertheless, a reference will never have more than one entity link for each entity. Our UML diagram with entities added is shown in Figure 2.5.
2.3.5 Super Groups

In order to efficiently determine related entities, entities are divided into super groups. A **super group** is formed when references from two different entities share the same closure attribute (discussed later). In other words, super groups are the connected components in the closure graph described in section 2.3.3. Each entity instance (represented by an entity link) will belong to exactly one super group. Furthermore, a super group only contains entities of the same type (e.g., Consumer or Household). Extending our UML diagram with super groups leads to the Figure 2.6. While other authors have discussed the role of transitive closure in entity resolution (Singla and Domingos 2006), this is the first instance of a term to denote the structure formed after closure occurs. An interesting use of super groups is shown in Bhattacharya and Getoor (2005) where the authors mine latent similarity structures in the closure graph of a super group in order to find unspecified associations.
2.3.6 Matching Graphs

To maintain information on the connections between managed references, entities use a matching graph. A **matching graph** is a graph that contains a vertex for each managed reference associated with the entity and an edge for each positive association between managed references. For instance, “Bob Smith 123 Main St. 21543,” “Robert Smith 123 Main St. 21543,” and “Bob Smith 18 East Dr. 93253” are three different managed references that belong to the same entity. The first two are connected by an edge that represents a fuzzy match. The last might be connected to the first by external information. For instance, the edge might represent the fact that Bob Smith has moved and has filled out a change of address form. The resultant matching graph for the entity is visually represented in Figure 2.7.

There is a one-to-one correspondence between matching graphs and super groups; therefore, there are typically many entities participating in one matching graph. The difference between the matching graph just described and the closure graph described in section 2.3.3 needs to be stressed. The closure graph is used to find which entities participate in the matching graph. The edges on the closure graph directly correspond to identical closure attributes. The vertices on the closure graph correspond to managed references. The matching graph is used to determine how managed references are organized into entities. The vertices again refer to managed references, but the edges refer to fuzzy or logical associations between references. So, the vertices between a closure graph and a matching graph will be identical, but the edges will be very different.
Fig. 2.7. Bob Smith and Robert Smith represent the same consumer entity. Two references have been joined by a fuzzy matching algorithm. The other two references have been joined by a change of address processed from post office data.

The nouns presented in this section form the foundation of the taxonomy. In the next section we will augment these nouns with adjectives and in section 2.5 we will give them actions via verbs.

2.4 A Declarative Taxonomy: The Adjectives

Some nouns from the previous section can take different forms and therefore may have adjectives attached to them. Specifically, attributes and references can be augmented in the system by descriptive terms.

2.4.1 Attribute Adjectives

An attribute is a very flexible and vague piece of information; therefore, it stands to reason that it may have the most adjectives defined to clarify its purpose. Some attributes with little relative importance, such as a street pre-directional (N, S, E, W), may have no adjectives assigned to them. Others with more importance, such as a name, may have multiple adjectives. All of the attribute adjectives are orthogonal to one another and may be used simultaneously to describe an attribute. The attribute adjectives defined by our taxonomy include the following
Search key: An attribute that is used in its raw form by some path’s prospector. An account number or other uniquely identifying piece of information will typically have this modifier.

Indirect search key: An attribute that is processed or combined with other attributes before being used by a prospector. An attribute may be used both as a search key and as an indirect search key.

Closure attribute: An attribute that is used in its raw form to formulate closure for an entity. An attribute may contribute to more than one entity’s closure. For example, a phone number may be a valid closure attribute for a consumer entity as well as a household entity.

Indirect closure attribute: An attribute that is processed or combined with other attributes before contributing to an entity’s closure. A good example might be certain address attributes (street and zip) being combined into a fuzzy address key which is used by the household entity to formulate closure.

Identity attribute: Two references are identical if all of their identity attributes are equal. Identity attributes help to prevent duplicate references being persisted. Typically, more distinguishable attributes such as first-name, last-name and street are identity attributes. Attributes that are typically not identity attributes include title and gender.

Implied attribute: An entity uses implied attributes to prevent invalid reference combinations. For instance, we might wish to prevent a reference with a name suffix of SR from being in the same entity as a reference with a name suffix of JR. In that case, we would say that name suffix is an implied attribute. Once an attribute is designated implied then all references that belong to an entity must have the same attribute value (or no value at all for that attribute).

Constrained attribute: An attribute that has certain business rules around its format. For example, the entity resolution system might wish to store name suffix values as the integers 1 to 9. If this were the case, then the name suffix field is said to be constrained. Other constraints might include all digits for an account number, a certain format for a phone or social security number, or a birth date field matching a given regular expression.

Nullable attribute: A given reference type may decide that certain attributes are so important that they cannot go missing. These attributes are non-nullable. A nullable attribute may be either present or absent without any ill effects. If an input reference is missing a non-nullable attribute, it may still receive a link from a managed reference, but it will never be persisted.
2.4.2 Reference Adjectives

Adjectives that augment references do so to describe how references relate to entities. Does a reference type exist to be grouped into various entities, or is the reference type only there to help conceptualize the domain model? The adjectives that answer those questions are

**Linkable:** A linkable reference is a managed reference that has been assigned a unique id, known as a link. The entity resolution system will assign that link to any input reference that matches the managed reference. These links may be used to later update or delete references that are in need of maintenance.

**Non-linkable:** A non-linkable reference is purely logical and has no linkage. For instance, we can think of a Person reference as consisting of a first-name, middle-name, last-name, name-suffix, primary-number, street, secondary-number, and zip. Or, we can think of a Person reference consisting of a Name reference and an Address reference. These references are not members of other entities; therefore, we don’t need to manage them individually and we don’t assign them links. They are present only to help conceptualize the system.

2.5 The Declarative Taxonomy: The Verbs

An entity resolution system must also exhibit actions. These actions can be configurable in many different ways, thus the taxonomy defines verbs that are often implemented as pluggable strategies rather than as hard-coded modules. Verbs can be applied at the attribute level, reference level, or entity level.

2.5.1 Attribute Verbs

Since the attributes are the atomic data elements, attribute verbs describe various manipulations that may be performed on the data. The verbs are

**Standardize:** This action mutates an attribute to ensure consistency and adherence to database rules. For instance, all input fields might be trimmed and certain illegal characters (‘, -) might be removed. Standardization for US name suffixes might entail translating I and SR to 1, II and JR to 2, and
so on. Standardization might also be used to ensure only certain values are allowed. For instance, if a first-name attribute consisted of only numbers, then it might be considered invalid and changed to blanks. Because of possibility of mutation, standardization must occur before anything else that uses the input references (e.g., path selection, prospecting, or matching). Furthermore, it is important to distinguish between standardization, which merely ensures database constraints, and hygiene, which is more encompassing in nature. For instance, address hygiene is an important part of data cleansing, but would be too involved for standardization. Though the line is a fine one, the important decision factor is whether or not the rule enforces a database constraint. If so, it belongs in standardization. If not, it belongs outside the entity resolution system.

**Validation:** A subset of standardization in which an attribute is either accepted or rejected. If the attribute is rejected, the attribute may be removed (replaced with blanks), or the entire reference may be removed from the input source or the reference have its insertion prohibited. Validation ensures that attributes exist, only contain certain characters or values, or only occur in conjunction with other attributes. For instance, a primary number might only exist when the street is not a PO Box.

### 2.5.2 Reference Verbs

Actions involving references revolve around reference management and linkage. References are independent of entities, so the reference actions do not involve their associations with entities. The reference verbs are

**Match:** The most basic question that can be asked is whether or not two references are equivalent. To ask this question programmatically, the reference match function is used. At a minimum, the match function should return back whether the references are close enough to be considered the same reference. A more useful function will return additional information such as the strength of match. In general, match functions can be divided into two types. **Attribute-aware** match functions have knowledge about the attributes that compose the reference. Their usefulness comes from their ability to use the attribute type to make a better match. For instance, if the match function knows that a reference has a first name it can use a first name nickname algorithm to provide a more accurate match. **Attribute-neutral** match functions do not have special knowledge of a reference’s attributes. Their usefulness comes from their ability to work regardless of locale or reference genre. Often, an attribute-neutral match function will work equally well across languages or even on something as diverse as barcodes.
Create: To manage an input reference, we must create a managed reference in our database. The creation function will need to generate a unique reference link for the new reference.

Remove: For various reasons, references will need to be removed. Occasionally, the provider of the managed reference will request that their information be removed from a database. Another removal reason would be an incorrect entity resolution run. Removal of a reference may also affect the structuring of entities, so analyzing the matching graph after removal is a necessity.

Update: Often, a reference will need to be updated due to inaccurate or incomplete information. For instance, a customer may call in to say that his phone number or email address is incorrect, or a government might change the postal code or street address for a particular person. Like the remove operation, update may affect the entity structure and requires the matching graphs to be reanalyzed.

Reconcile: Reference reconciliation is a strategy for deciding when to create or update a reference. It reconciles new information with the information that is already known. A reconciliation strategy should use the proximity of match between the input and managed references as well as the data provenance to determine what action needs to be taken. A common strategy might include doing nothing if the input reference is a subset of the managed reference. However, if the managed reference is a proper subset of the input reference, then the managed reference might be updated. Furthermore, the reconciliation strategy may take into account information about the source of the input or managed reference. For example, if the input reference was taken from a marketing file, it may not be allowed to update a managed reference that came from an internal account file. Moreover, some sources should never be added to the database for legal reasons, so the reconciliation strategy should refuse to insert records from those sources. Therefore, reconciliation is a way to ensure the entity resolution system treats sources as first class citizens with regards to its persistence policies.
Fig. 2.8. An example reconciliation algorithm

2.5.3 Entity Verbs

Entities are more volatile than references, because they may be organized and re-organized often. Therefore, the taxonomy defines entity verbs that concern themselves with the merging and unmerging of entities as well as with determining membership for references. The verbs defined for entities are

**Includes-reference:** This verb decides if an entity should contain a given reference. This takes into account the entity match rules as well as the implied attributes of the references. Typically, to determine membership, the candidate reference is compared to each reference in the entity. If the entity match rules ever return a positive match, then the candidate reference should be included (assuming no implied field conflicts); otherwise, it should not.

**Split:** A split occurs when one reference needs to change entities. An input reference is added to its first entity by splitting it from the null entity. When a
managed reference is deleted, it is split from its current entity to the null entity. Splits do not affect closure and should not affect the matching graph, since it should have already been updated, thus causing the split.

**Reconcile:** If the entity needs to include a new reference or split out an old reference, then it should invoke the reconciliation action. This action is designed to preserve the integrity of the database when presented with the myriad of changes that a split or new reference can entail. First, the database needs to be locked against any changes to the closure (since we’ll be updating the matching graph). The lock needs to extend to any references that will be split. Second, the references being split need to have their group links updated to the new group link. Third, the matching graph needs to be updated and finally, the records need to be unlocked.

Entity reconciliation is also responsible for reconciling super groups. If a reference is added whose closure attributes correspond to more than one closure key, then those super groups must be consolidated. This occurs in two steps. First, the matching graphs for the various super groups are joined together by use of the entity match rules. Second, the lowest closure key is chosen as the new closure key. Third, all references with the other closure keys will have their closure key changed to match the lowest. Fourth, the consolidated closure keys’ matching graphs will be removed from the base and the new matching graph will be added and associated with the lowest closure key. A possible entity reconciliation algorithm is shown in Figure 2.9.

```plaintext
Lock appropriate matching graphs and included occupancies
Update managed references with new group links
If (input reference spans super groups)
    lowestClosureKey := min(input reference closure keys)
    update managed references, set closure key to lowestClosureKey
    delete matching graphs with closure key ! = lowestClosureKey
End if
Update matching graph
Unlock all locks
```

**Fig. 2.9.** A possible entity reconciliation algorithm.

### 2.6 A Declarative Representation

A key factor in making a system declarative is the representation of the system. It must be focused on describing the data and relationships and not in describing the algorithms. For that reason, the XML is used as the representation language. The
XML is known for its flexibility and suitability for describing data, which is needed for the entity resolution system.

### 2.6.1 The XML Schema

Since attributes are the most atomic unit of the taxonomy, it makes sense to start by representing them. An attribute needs an identifier. This identifier should be unique across the system. An attribute may also have a maximum length, a standardization or validation routine, a nullable property, and whether or not it is an identity attribute for the enclosing reference. Other attribute adjectives are applied when describing other nouns. An XML snippet would look like Figure 2.10.

```xml
<attribute id="first-name"
  max-length="25"
  nullable="false"
  identity="true"
  validation="[A-Za-z ]+"/>
```

**Fig. 2.10.** The XML representation of an attribute. The validation routine is an inline regular expression ensuring the attribute only consists of alphabetic characters.

References are the next component of the system and are composed of attributes. Therefore, it seems natural to represent references as a composite element. In addition, references have paths, which are themselves composed of prospectors. The prospectors are constructed via a dependency injection framework such as Spring (Spring 2008); therefore, they may have configuration parameters in the declarative XML representation. A reference XML snippet is shown in Figure 2.11.
Fig. 2.11. A reference with two prospecting paths, name/address and phone.
Finally, the entity representation is shown in Figure 2.12. Entities need to be associated with a reference type, entity match rules, and closure attributes. The closure attributes need not be part of the reference type, they can be generated attributes as well. Furthermore, the implied attributes can be specified in the match rules, in the XML schema itself, or in both.

```xml
<entity id="Consumer"
    reference="PersonAtAddress"
    matchrules="consumer.matchrules">
    <closure-attributes>
        <closure-attribute ref="Phone"/>
        <closure-attribute
            generator="com.recognition.AddressKeyGenerator"/>
    </closure-attributes>
</entity>
```

**Fig. 2.12.** The XML for the *Consumer* entity.

### 2.6.2 A Representation for the Operations

The operations would typically be configured in a more dynamic way than XML due to their implementation as strategies. For instance, one could envisage managing a data source’s retention and update policies through a web interface. However, the declarative schema can be used for more static decisions such as whether or not to allow or disallow operations such as splits, deletions, or reference creation. For more control, operations such as update could be specified at the attribute level. Figure 2.13 shows such a representation.

```xml
<reference id="PersonAtAddress" …>
 ...
<operation id="delete" status="disabled"/>
<operation id="update">
    <legal-attributes>
        <legal-attribute ref="Phone"/>
        <legal-attribute ref="Zip"/>
        <legal-attribute ref="Street"/>
    </legal-attributes>
</operation>
</reference>
```

**Fig. 2.13.** Static decisions such as whether or not to allow deletes and which attributes may be updated should be placed in the declarative XML.
2.7 Conclusion

In this chapter, the entity resolution problem is described and defended. In addition, we have proposed a declarative taxonomy for defining an entity resolution system and laid the foundation of a declarative language based on the XML. The contributions include a consistent taxonomy for entity resolution including nouns, adjectives and verbs. Moreover, the taxonomy is structured in such a way that it promotes pluggable strategies, enabling the re-use of other author’s research. Finally, the XML is used to apply our taxonomy to an actual implementation. The declarative approach in conjunction with the XML definition will facilitate easier entity resolution system deployment and may make the data integration problem more agile. The author’s team at Acxiom Corporation is currently investigating this approach to specifying an entity resolution system. Java and J2EE are being used to allow for pluggable strategies in the form of JavaBeans. So far, the results have been very positive with the ability to define entity resolution systems in a matter of hours instead of days or weeks with the current ETL based approach. Future research into this area could concentrate on integrating data provenance policies (when to insert, update, and delete based on sources), further refining the path-based prospecting approach, and positive or negative associations between entities for law enforcement (who bought guns from dealer X and also lived with known suspect Y).

2.8 Exercises

1. Describe how your school or workplace might use entity resolution.
2. Name the attribute adjectives. For each adjective, tell whether its meaning has the most effect on the attribute itself, the reference, or the entity.
3. Would a postal (or zip) code make an effective closure attribute? Why or why not? How could you modify the zip code to make it more attractive as a closure attribute?
4. Maintaining associations between entities are critical for law enforcement agencies. How would you extend this language to handle those associations? In addition, negative associations are also important. These associations ensure that two references never belong to the same entity. How do positive and negative associations affect the verbs of our taxonomy? How do they affect the XML representation?

2.9 References

Data Engineering
Mining, Information and Intelligence
Chan, Y.; Talburt, J.; Talley, T.M. (Eds.)
2010, XVII, 447 p., Hardcover
ISBN: 978-1-4419-0175-0