DESIGN CONSIDERATION FOR IDENTITY RESOLUTION IN BATCH AND INTERACTIVE ARCHITECTURES

(Research-in-Progress)

Fumiko Kobayashi
Department of Information Science, University of Arkansas, Little Rock, AR, USA
fxkobayashi@ualr.edu

Eric D. Nelson
Department of Information Science, University of Arkansas, Little Rock, AR, USA
ednelson@ualr.edu

John R. Talburt
Department of Information Science, University of Arkansas, Little Rock, AR, USA
jrtalburt@ualr.edu

Abstract: This paper discusses the implications of running an identity resolution (IR) process in interactive mode versus batch mode. In an IR process, entity references are resolved against a known set of identities (master records). In traditional batch mode, the final resolution decision for each input reference is postponed until after all references have been read and processed. This makes it possible for a reference read later in the input sequence to influence, and sometime change, the resolution decision of references read before it. When IR systems are called upon to run in online interactive environments, the resolution decision must be made in real-time as each reference is received. The result is that the same IR system processing the same sequence of input reference may give different (inconsistent) identity resolution decisions. The paper also outlines some of the conditions that cause batch and interactive IR systems to give different results and proposes designs for interactive systems that can mitigate this problem.

Key Words: Entity Resolution, Identity Resolution, OYSTER, Information Quality, Batch architecture, Interactive architecture

I. BACKGROUND

a. Entity Resolution

Entity Resolution (ER) is the process of determining whether two references to real-world objects are referring to the same object or to different objects [9]. Real-world object can be identified by their attributes and by their relationships with other entities. Most Entity Resolution systems take entity references and base their decisions upon the degree to which two references have similar attribute values, i.e. "matching" to make this decision. Attributes vary depending on the type of entity being described; some examples of attributes are First Name, Last Name, and Social Security Number for references about people or Car Make, Car Model, and VIN for references about cars. ER has also been studied under other names including but not limited to record linkage [3], deduplication [7], reference reconciliation [10], and object identification [8].

Entity resolution systems can be categorized into four architectures [9]:

- Merge-Purge
- Heterogeneous database join
- Identity Resolution
- Identity Capture

This paper focuses on design issues related to Identity Resolution architectures.

b. Identity Resolution

In an identity resolution (IR) process each entity reference input into the system is resolved against a previously defined set of identities represented as some type of entity identity structure (EIS). When the pre-defined identities represent master data of an organization such as employees, customers, or products, IR becomes an important component of master data management (MDM). MDM and IR in a broad set of applications including business, health care, education, law enforcement, and the military. In a business context of MDM of customer information, IR is sometimes called "customer recognition" [9].

In most IR systems, identity resolution decisions are based on a combination to two things, a set of one or more matching rules and the transitivity of resolution.

Transitivity refers to the transitive relationship between entity references. Put simply, it means that if entity reference A and entity reference B are equivalent, and entity reference B and entity reference C are equivalent, then reference A is equivalent to reference C and all three references (A, B, and C) refer to the same real-world entity. In ER, there is a unique reference assumption that states every entity reference is created to refer to one, and only one, entity. It is this assumption that allows IR systems to depend on transitivity for their resolutions.

An example of transitivity can be seen by applying the following match rules to the references defined below.

- Match Rules
  - Rule 1: name and street must match
  - Rule 2: name and phone must match
- Entity references
  - A = (Mary Smith, 123 Oak, 555-1234)
  - B = (Mary Smith, 456 Elm, 555-1234)
  - C = (Mary Smith, 456 Elm, null)

Applying the match rules it is found that A matches B by Rule 2 and that B matches C by Rule 1. A does not directly match C through any rule, but C is in transitive closure of A. This means that through a transitive relationship all three entity references (A, B, and C) refer to the same real world entity.

II. PROBLEM DEFINITION

Historically, IR has been run in batch mode where all identity resolution decisions are pending until the last reference in the batch is processed. In this way there is more information on which to base the final decision for reference. When processing a reference in a batch of references, the IR system makes a preliminary decision that the reference resolves to one of the known identities or that it references some other entity not represented in the system. However the decision that the reference does not resolve to one of the known identities may change as subsequent references are processed. This happens because

identity resolution decisions must be transitive in order to give consistent results independent of the order of processing.

The example defined in the previous section can be modified to provide a clear understanding of what takes place during the IR process. The order in which the entity references are processed will now be:

- Entity References
  - A = (Mary Smith, 123 Oak, 555-1234)
  - C = (Mary Smith, 456 Elm, null)
  - B = (Mary Smith, 456 Elm, 555-1234)

By providing this new order the first step performed is to try to match references A and C with match rules 1 and 2. Since the two references do not match on either rule, the system will decide that both A and C refer to two different entities and assign them separate identifiers. These are the initial decisions created by the IR system. The next step performed is to read reference B and match it against reference A and C. The IR system finds that A and B are equivalent based on rule 2 and B and C are equivalent on rule 1. The system then treats reference B as a glue record that is used to bring references A and C together so that the final decision shows A, B, and C all referring to the same real world identity. Through this process it can clearly be seen how future references affected the initial decisions by providing the system with additional information it did not previously have.

When an IR process runs in interactive mode, the user expects an identity decision (response) as each reference is submitted. In this case the preliminary decision is the final decision from the standpoint of the user. The system can be passed multiple references but each one is processed one at a time as its own batch. The decision is made and an answer is given back to the user without waiting for any additional information to be gleaned from following references. Once the information is given back to the user for the first reference, there is no opportunity to use the data provided by the trailing references that may impact the original decision and the original response cannot be changed. In an Interactive architecture the results are given in real time making impractical to exploit any information provided by following references.

This paper discusses under which situations a simple look-up is advantageous and when a more complex update must be performed before the look-up operation is initiated.

III. BATCH MODE

Batch Architecture is one of the simplest architectures that exist [1]. In most situations a program or solution that operates in this mode can, in a logical sense, only operate locally. In a physical sense, the process can actually operate on a remote machine many miles away. These systems typically also operate on files containing many references instead of single references. The Identity Resolution system starts by reading the entire repository, the set of master records, into memory. As each source input reference is read in it is merged with any identities from the identity structure that are found to match. This updates the repository which allows for transitive closure on identities where matches occur. Once the updates to the identity repository are completed, then the look-up can be performed and the corresponding IDs are returned to the user and the updated repository held in memory will be discarded. Since batch modes are typically localized any changes to the repository would not affect any other run because the repository would be reloaded every time the system is started. An IR system that runs in batch mode could work similar to the diagram in Figure 1.

The updated IDs provided will match closely with identities produced when performing Entity Resolution through the use of Identity Capture. Identity Capture is the method by which the repository can be updated with new information discovered from the input references. It can be argued that another need for Identity Resolution with Update arises from a need for consistency between the output of Identity Resolution and Identity Capture architectures.

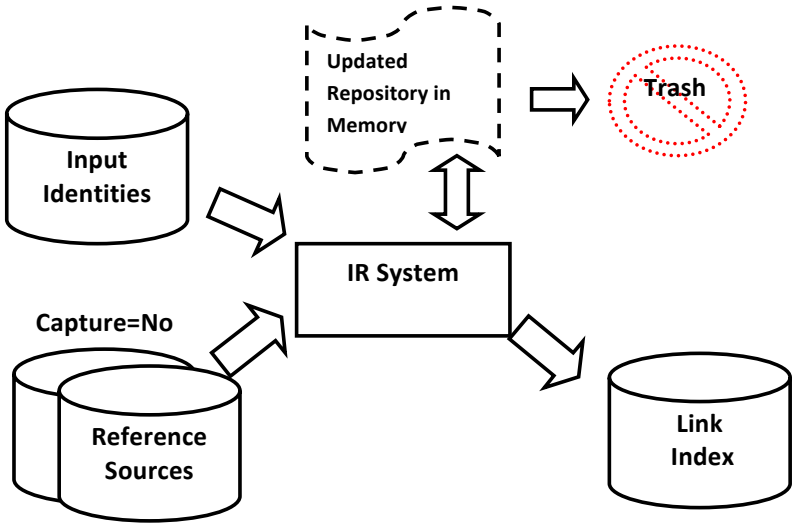

Figure 1. Identity Resolution with Update

There are several pros and cons to an IR system that is built on a batch architecture:

- Pros
  - Performs Transitive Closure of identities over many input references
  - Identities closely resemble Identities generated by Identity Capture Architecture.
  - Provides an additional method of validation
  - Represents information about the reference by returning an identity that encapsulates all known knowledge associated to the reference.
- Cons
  - Requires additional processing to perform updates
  - Can cause look-up issues for other users if performed in an Interactive environment since the identities returned may not match the identities in the repository.

IV. INTERACTIVE MODE

Interactive Architectures are often more complex. They typically operate more in a logically global sense. A platform-independent interface is used which implies that a client from anywhere, on any OS, and in any language, can consume the service [2].These systems can be stateless or stateful [5]. The user's state can be input, any configuration, and internal memory. IR systems in this architecture typically work on single references rather than references in bulk. If multiple input references are passed to the system as a single batch, the system will divide the references and treat each one as its own batch. This allows for the results of each reference to be passed back to the user as each reference is processed. An Interactive Identity Resolution system would work in the fashion depicted in figure 2. It starts by loading the entire repository into memory and then waits for a user request; this is only done once when the service is

initially started. As each users request is received a look-up is performed and a resolved ID is returned to the user.

There are several pros and cons to an IR system that is built on an interactive architecture:

- Pros
  - Acknowledges if a reference is recognized in the repository or not
  - Less processing required to resolve since update is not performed
  - Faster
  - Allows multiple users to request IR from the same repository at the same time
- Cons
  - Does not return all matches, only the first.
  - Does not represent all known information about the reference.
  - Not able to exploit any information provided by following references when making decisions

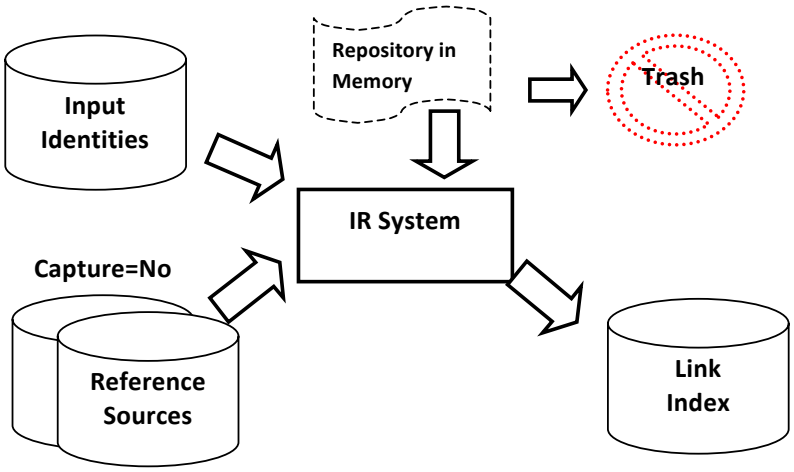

Figure 2. Identity Resolution with Look-up

V. INTERACTIVE VERSUS BATCH

It was initially believed that the Identity Resolution regardless of the systems architecture was simply a method in which to perform look-ups of the source references against a previously built identity structure. In test IR systems it was noticed that look-ups were not always sufficient to find the complete answer that was being asked, which essentially is, what is the identity of reference ***X***? In Interactive architectures, direct look-up operations provide a quick and easy method to recognize an identity of X, i.e. is there an identity in the repository that corresponds to reference X, yes or no.

From a functionality standpoint, in an Interactive mode, it is more suitable to simply return the ID associated to the identity that the reference resolves to and not try to perform any update. However, if the user wants to know more than if the reference simply exists then Interactive mode is not suitable since it only returns a single reference even when multiple identities may fulfill the requirements defined by the match rules. Also, if the input reference lacks the attributes necessary to resolve the reference, Interactive mode can fail and as a result return no identity. The primary reason that you would not want to update the Identity Structure is that if every input reference changed the repository, stored in memory, it would not be long before the repository would be unusable. A reference submitted by User A should not influence

the recognition of a reference submitted by User B especially since the quality or integrity of the sources of these might not be trusted. Here there would be no modification of the Repository since these matches should be transitory in nature.

In a Batch architecture, a more complete picture can be developed by performing the updates prior to look-ups. As previously mentioned, all resolutions are pending until the last reference in the batch is processed so that there is more information on which to base the final decision. From a performance stand point, performing updates to the repository structure prior to performing the look-up would introduce additional processing requirements and would in turn extend the time required to present results to the user. This issue of granularity is one of the differences between the Batch architecture and the Interactive architecture. In Batch mode, 10,000 references are processed prior to making the 10,000 resolutions where as in an Interactive architecture there is no preprocessing, the 10,000 references are resolved one at a time in the order they are received.

The differences in how the references are resolved provide room for discrepancies in the resolutions (as demonstrated in later sections) between the two methods. This discrepancy in results is one of the main issues that need to be addressed when building the two separate architectures. Questions such as "Should updates be allowed in Interactive architectures" and "How do we notify previous users that the Identifier provided for their entity reference has been updated or changed?" should be addressed and suggestions are provided in the following sections.

VI. Proposed Solutions for IR Architecture

The following outlines four proposed solutions that can be used for IR processing.

a. Look-up Solution

A look-up solution would involve a system that does not perform any form of updates to the repository. The IR system would only perform a simple look-up operation of comparing an incoming reference directly against the list of pre-defined identity structures. This solution would ignore any knowledge gained from interactions between current or future references. This type of solution has the advantage of producing consistent results (i.e. same entity identifier for the same reference) as long as the repository remains unchanged. The major drawback to this type of system is that it discards any new information that may be contained in the references being processed. This new information may be important and could help the system to make more accurate resolution decisions. At the same time, the look-up solution provides fast response time since it can build and maintain a static look-up index.

b. Progressive Solution

A progressive solution is an IR system that performs look-ups and returns an entity identifier to the user in real-time while retaining any new information gained in processing reference for use in process future references. This type of system will capture information from references to improve future decisions but does not have the ability to revise early decisions. Essentially, it allows later references to benefit from early references. A drawback of this system is that it can be inconsistent. If the system processes Reference 1 and returns the entity identifier A, then it is possible that a later reference could provide information that would change the entity identifier associated for Reference 1. If the same Reference 1 were processed again, then the system could return a different entity identifier (B) due to the change effected by the new information.

c. Notification Solution

A notification solution is similar to the progressive solution in that it returns the requested information to the user while retaining the information to use for future transactions. The difference between the two solutions is that the notification solution will keep a catalog of all references processed by the system and the entity identifier that was returned to the user. In the case that new information causes a previously published identifier to change, a notification can be sent to user with the new entity identifier value. The notification solution assumes that users provide enough information to the system that any changes can be properly communicated. The notification system model demands additional design considerations as to how long of a processed reference should be maintained. Is it important to notify users that their reference processed two years or ten year ago has changed? The answer will depend on the type of data that is being processed and maintained by the system and the business rules in place at the organization making use of the system.

Both the Progressive and the Notification Solutions may also be limited by contractual or legal condition related to the use and retention of information provided by the user.

d. Transient Repository Solution

In a Transient Repository solution, an input reference is resolved against a transient repository of all potential candidates. When a request is made, all the identities from the repository that are potential candidates to match the source reference are pulled into their own memory space creating a separate transient repository of identities. The system then performs resolution for the input references against this transient repository to update the identities therein with the new information for this single request. The identities to be included as candidates in the transient repository are defined by finding all identities in the full repository that have non-deterministic matches to the incoming source reference(s). The system can then perform a look-up on the reference against the transient repository and the appropriate result would be returned. Once the result is returned the memory space for the transient repository is released, effectively discarding the transient repository. This solution will provide a more informative result than the look-up solution since the returned identity represents all knowledge about the reference that is currently contained in the full repository. Since the new information is discarded, all future iterations of the same entity reference will undergo the same resolutions against the same set of identities allowing this method to provide consistent results. One drawback of this solution is that it will increase the processing time of the Interactive architecture as it would introduce both a form of transitive closure and a preprocessing method to find all possible matches. Another drawback is that since the system uses a transient repository, it does not retain any of the information for later use and all gained information is not integrated back into the main repository.

VII. OYSTER

OYSTER (Open sYSTem Entity Resolution) is an open source project sponsored by Center for Advance Research in Entity Resolution and Information Quality (ERIQ) and the University of Arkansas at Little Rock [9]. OYSTER was originally designed to support instruction and research in ER by allowing users to configure its entire operation through XML scripts executed at run-time. The resolution engine of the current version (3.1) supports probabilistic direct matching, transitive linking, and asserted linking [11]. OYSTER builds and maintains an in-memory repository of attribute values to identities that allows it to manage identities quickly and efficiently. Because OYSTER has an identity management system, it also supports persistent identity identifiers.

OYSTER is driven by multiple XML scripts that tell the system where to locate the input repository, how the input source files are structured, and what match rules should be used to determine matches. OYSTER can be configured, through these XML files, to handle 3 of the 4 recognized architectures of ER:

- Merge-Purge
- Identity Resolution
- Identity Capture

It also can be configured to handle Assertions, which is a way to do assertive linking of references.

One of the most noticeable features of OYSTER is its ability to handle an Identity Update architecture which effectively allows OYSTER to combine Identity Capture and Identity Resolution into a single run. This allows for a repository of identities to be defined and maintained.

OYSTER source code and documentation are freely available from the ERIQ website (http://ualr.edu/eriq).

a. Practical Approach with OYSTER

This example shows the difference in results when Identity Resolution is run interactively as a service with look-up compared to running it in batch with updates. This example and the results are produced by OYSTER. The data used is made for the purpose of this example and is not actual real world data. Currently OYSTER only performs Identity Resolution with Look-up but will be implementing IR with updates in the 3.2 release. Due to the proposed extension of the Identity Resolution architecture not being implemented in the current version of OYSTER, the Identity Resolution with Update results were produced by modifying the output of an Identity Capture run since the only difference in the outputs should be the references that do not match any identities.

For the examples, a test repository was constructed (Figure 3) that will allow for a concise and accurate demonstration.

```
<root>
	<Metadata>
		<Modification OysterVersion="3.1" Date="2011-07-02 16.10.14" RunScript="null" />
		<Attributes>
			<Attribute Name="@RefID" Tag="A"/>
			<Attribute Name="StudentFirstName" Tag="B"/>
			<Attribute Name="StudentLastName" Tag="C"/>
			<Attribute Name="SocialSecurityNbr" Tag="D"/>
		</Attributes>
	</Metadata>
	<Identities>
	<Identity Identifier="IX9QEXQOG4IE5EAL" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.37|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
		</References>
	</Identity>
	<Identity Identifier="S9BPXS63M657EP80" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.110|B^MIKE|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	<Identity Identifier="TIHKPSPFVTFBAEQN" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.10|B^MICHAEL|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	</Identities>
</root>
```

Figure 3. Input Identity Repository

Figure 4 shows the new input references that OYSTER will resolve against existing identities in the repository.

```
IdentityID|FirstName|LastName|SSN
210|MIKE|DANIELS|010-64-1563
555|TAMMY|KRUMP|023-99-9901
137|MICHAEL|AMPEREZ|010-89-4193
```

Figure 4. Input references

This example uses two match rules (Figure 5), rule 1 and 2. Each rule will match reference 210 to a different identity in the repository.

```
<Rule Ident="1">
    <Term Item="StudentFirstName" MatchResult="Exact"/>
    <Term Item="StudentLastName" MatchResult="Exact"/>
    <Term Item="SocialSecurityNbr" MatchResult="Exact"/>
</Rule>
<Rule Ident="2">
    <Term Item="StudentFirstName" MatchResult="NICKNAME"/>
    <Term Item="StudentLastName" MatchResult="Exact"/>
    <Term Item="SocialSecurityNbr" MatchResult="Exact"/>
</Rule>
```

Figure 5. OYSTER Identity Resolution Rules

b. Identity Resolution in a Batch Architecture

When configured to perform Identity Resolution in a Batch architecture, updates can be made to the repository that is loaded into memory. OYSTER starts by reading the repository into memory. Next, OYSTER starts merging the references from the source into the repository. Matching on rule 1 will provide the Repository (Figure 6) to be updated in memory.

```
<root>
	<Metadata>
		<Modification OysterVersion="3.1" Date="2011-07-02 16.10.14" RunScript="null" />
		<Attributes>
			<Attribute Name="@RefID" Tag="A"/>
			<Attribute Name="StudentFirstName" Tag="B"/>
			<Attribute Name="StudentLastName" Tag="C"/>
			<Attribute Name="SocialSecurityNbr" Tag="D"/>
		</Attributes>
	</Metadata>
	<Identities>
	<Identity Identifier="IX9QEXQOG4IE5EAL" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.37|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
			<Reference Value="A^source2.137|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
		</References>
	</Identity>
	<Identity Identifier="S9BPXS63M657EP80" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.110|B^MIKE|C^DANIELS|D^010-64-1563"/>
			<Reference Value="A^source2.210|B^MIKE|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	<Identity Identifier="TIHKPSPFVTFBAEQN" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.10|B^MICHAEL|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	</Identities>
</root>
```

Figure 6. In-memory repository after first rule match to reference 1

Once the first match rule completes, the second match rule is used and produces the updated repository in memory (Figure 7).

```
<root>
	<Metadata>
		<Modification OysterVersion="3.1" Date="2011-07-02 16.10.14" RunScript="null" />
		<Attributes>
			<Attribute Name="@RefID" Tag="A"/>
			<Attribute Name="StudentFirstName" Tag="B"/>
			<Attribute Name="StudentLastName" Tag="C"/>
			<Attribute Name="SocialSecurityNbr" Tag="D"/>
		</Attributes>
	</Metadata>
	<Identities>
	<Identity Identifier="IX9QEXQOG4IE5EAL" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.37|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
			<Reference Value="A^source2.137|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
		</References>
	</Identity>
	<Identity Identifier="S9BPXS63M657EP80" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.110|B^MIKE|C^DANIELS|D^010-64-1563"/>
			<Reference Value="A^source2.210|B^MIKE|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	<Identity Identifier="TIHKPSPFVTFBAEQN" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.10|B^MICHAEL|C^DANIELS|D^010-64-1563"/>
			<Reference Value="A^source2.210|B^MIKE|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	</Identities>
```

Figure 7. In-memory repository after second rule match to reference 1

Once all match rules are completed, transitive closure can be performed on references 1 and 2 and since *110 =210,* and *10 = 210,* then *110 = 10.* The final repository will look like Figure 8. References that cause transitive closure to be performed on existing identities in a repository are called the glue references.

```
<root>
	<Metadata>
		<Modification OysterVersion="3.1" Date="2011-07-02 16.27.41" RunScript="null" />
		<Modification OysterVersion="3.1" Date="2011-07-02 16.10.14" RunScript="null" />
		<Attributes>
			<Attribute Name="@RefID" Tag="A"/>
			<Attribute Name="StudentFirstName" Tag="B"/>
			<Attribute Name="StudentLastName" Tag="C"/>
			<Attribute Name="SocialSecurityNbr" Tag="D"/>
		</Attributes>
	</Metadata>
	<Identities>
	<Identity Identifier="IX9QEXQOG4IE5EAL" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.37|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
			<Reference Value="A^source2.137|B^MICHAEL|C^AMPEREZ|D^010-89-4193"/>
		</References>
	</Identity>
	<Identity Identifier="TIHKPSPFVTFBAEQN" CDate="2011-07-02">
		<References>
			<Reference Value="A^source1.10|B^MICHAEL|C^DANIELS|D^010-64-1563"/>
			<Reference Value="A^source1.110|B^MIKE|C^DANIELS|D^010-64-1563"/>
			<Reference Value="A^source2.210|B^MIKE|C^DANIELS|D^010-64-1563"/>
		</References>
	</Identity>
	</Identities>
</root>
```

Figure 8. In-memory repository after transitive Closure

Once the update is completed (Fig 8) it is apparent that reference 210 from the source should return the OYSTER ID that is assigned to the identity that is built from entity references 10, 110, and 210 in the updated repository. It is also apparent that reference 555 should be assigned an ID of XXXXXXXXXXXXXXXX since the reference does not appear in the updated repository that resides in memory. When performing Identity Resolution with Update the information can be used to determine possible impacts on the repository. The results from the run, like Identity Resolution with Look-up, can be used to verify a reference exists in the repository. This method requires more processing than running Identity Resolution with Look-up only since updates to the repository are being made prior to the look-up. Figure 9 shows the output from OYSTER after the resolution is completed.

```
RefID        OysterID            Rule
source2.555  XXXXXXXXXXXXXXXX    null
source2.210  TIHKPSPFVTFBAEQN      [1]
source2.137  IX9QEXQOG4IE5EAL      [1]
```

Figure 9. Identity Resolution with Update Link file after OYSTER run

Even though rule 2 is used during the update, it is not in the link index since OYSTER is designed to stop searching once the first match is met for the look-up part of Identity Resolution with Update. How the look-up part of the update should be performed is open to interpretation and depends on what information is needed, such as possibly supplying all match rules that a source reference uses to match to an identity.

c. Identity Resolution in an Interactive Architecture

This is similar to the activity performed by the current Identity Resolution architecture as implemented in OYSTER versions 3.1 and earlier. The only difference is that in this example, no link index is generated and the references are processed individually as they are received. When configured to perform Identity Resolution in an Interactive architecture a look-up only approach should be used. With this approach it is apparent that reference 137 from the source (Fig 4) should return the OYSTER ID IX9QEXQOG4IE5EAL (Fig 3) because of rule 1. It is also apparent that OYSTER should return an ID of XXXXXXXXXXXXXXXX for reference 555 since the reference does not appear in the repository. The OYSTER ID XXXXXXXXXXXXXXXX is used by OYSTER in Identity Resolution mode only. It signifies that OYSTER has never seen this entity reference before.

What is not clear is what OYSTER ID should be returned for entity reference 210. Since, as stated previously, there are two match rules, one that match 210 to identity S9BPXS63M657EP80 in the repository, and a second that matches 210 to identity TIHKPSPFVTFBAEQN in the repository. Typically, when performing Identity Resolution with Look-up, the most important information is if the reference exists in the repository or not. To increase performance, processing will abort on the reference once a match is found so that the entire repository does not have to be searched with every rule for a single source reference. This performance increase leads to reference 210 being paired with the identity S9BPXS63M657EP80 from the repository. Whether this is the way Identity Resolution should function in an Interactive environment is a matter for interpretation as to the goal of the look-up but this is how it was implemented in the OYSTER system. Figure 10 shows the output that is sent to the User during the resolution. The results are sent one at a time as Interactive mode treats each reference in a batch as its own batch.

```
RefID        OysterID            Rule
source2.210  S9BPXS63M657EP80    [1]

RefID        OysterID            Rule
source2.555  XXXXXXXXXXXXXXXX    null

RefID        OysterID            Rule
source2.137  IX9QEXQOG4IE5EAL    [1]
```

Figure 10. Identity Resolution Results in Interactive Architecture

Rule 2 is not mentioned in Figure 10 due to how OYSTER currently performs matching for Identity Resolution as explained previously.

d. Batch and Interacting Architecture Result Comparison

This example shows how, depending on the architecture, the Identities returned to the user can be different. It also shows how the Batch architecture allows all references in the batch to be processed prior to performing the look-ups on the references to resolve the IDs.

In the Batch architecture, the system did not perform any resolutions until all the update process was completed. Once the system integrated all references from the source into the repository, the system them proceeded to perform a look-up on each source reference and generate a single output file, called a Link Index, which is depicted again in Figure 11.

```
RefID        OysterID            Rule
source2.555  XXXXXXXXXXXXXXXX    null
source2.210  TIHKPSPFVTFBAEQN    [1]
source2.137  IX9QEXQOG4IE5EAL    [1]
```

Figure 11. Identity Resolution Batch architecture Link Index file after OYSTER run

The Interactive run accepted the same list of three references that were processed into the Batch run. The Interactive run divides the source input into three separate runs and processes each reference individually. After each reference is processed, the result is sent back to the user before the next reference is resolved. The results provided by the Interactive run are depicted again in Figure 12.

```
RefID        OysterID            Rule
source2.210  S9BPXS63M657EP80    [1]

RefID        OysterID            Rule
source2.555  XXXXXXXXXXXXXXXX    null

RefID        OysterID            Rule
source2.137  IX9QEXQOG4IE5EAL    [1]
```

Figure 12. Identity Resolution Results in Interactive Architecture

Due to the update performed in the Batch run, the OysterID returned for reference 210 is different than the ID returned to the user in the Interactive run. A few alternative methods were discussed earlier that address these discrepancies.

VIII. FUTURE WORK

Currently OYSTER is a batch architecture IR system. One future plan is to allow it to also work as an interactive architecture IR system. One task when this new version is released will be the testing and validation of the potential issues that can arise with IR look-ups. Another will be design strategies to help mediate these issues such as the side filing of potential references that can be used to update the repository and the use of temporary or on-the-fly recognition.

IX. CONCLUSION

In developing and using OYSTER the understanding of what is necessary for practical use beyond what is theorized [4][12] is constantly being redefined. IR systems that only perform look-ups can potentially be insufficient for some applications. This is due to the fact that look-ups do not provide a mechanism for transitive closure. The need for Identity Resolution to perform updates and there by transitive closure, which is also evident in certain situations, was outlined and discussed throughout this paper. It was shown that Batch and Interactive architectures influenced the type of IR that can be performed on references. In designing an IR system, careful thought should be given to what question the IR system will ultimately be answering as well as the underlying architecture to answer these questions. Four possible alternative Interactive systems were given that help to overcome some of problems found in updating an interactive repository. The architectures defined in this paper can provide good balance between consistent identification and system performance and complexity.

REFERENCES

[1] Ebbers, Mike. "7. Batch Processing and JES." Introduction to the New Mainframe: Z/OS Basics. *IBM, International Technical Support Organization*, 2009. 254-55. Print.

[2] Hashimi, Sayed. "Service-Oriented Architecture Explained." *O'REILLY Windows Devcenter. O'Reilly*, 18 Aug. 1983. Web. 14 July 2011. <http://ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html>.

[3] HB Newcombe, JM Kennedy, SJ Axford, and AP James. Automatic linkage of vital records. *Science*, 130:954–9, 1959.

[4] Nelson, E., & Talburt, J. (2011). Entity Resolution for Longitudinal Studies in Education using OYSTER. *The 2011 International Conference on Information and Knowledge Engineering (IKE'11).* Las Vegas, Nevada: (accepted for publication).

[5] Perrochon, Louis. (1994).Translation Servers: Gateways Between Stateless and Stateful Information Systems.In Proceedings of the Network Service Conference

[6] Purdom, P. A transitive closure algorithm, *BIT vol. 10* pp. 76-94, 1970

[7] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278. ACM New York, NY, USA, 2002.

[8] S. Tejada, C.A. Knoblock, and S. Minton. Learning object identification rules for information integration. Information Systems, 26(8):607–633, 2001.

[9] Talburt, J. R. (2011). *Entity Resolution and Information Quality.* Burlington, MA: Morgan Kaufmann.

[10] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96. ACM New York, NY, USA, 2005.

[11] Zhou, Y., & Talburt, J. (2011). The Role of Asserted Resolution in Entity Identity Management. *The 2011 International Conference on Information and Knowledge Engineering (IKE'11).* Las Vegas, Nevada: (accepted for publication).

[12] Zhou, Y., Talburt, J., and Nelson, E. (2011). The Interaction of Data, Data Structures, and Software in Entity Resolution Systems. *Software Quality Professional:* (accepted for publication)