iFuice – Information Fusion utilizing Instance Correspondences and Peer Mappings

Erhard Rahm, Andreas Thor, David Aumueller, Hong-Hai Do, Nick Golovin, Toralf Kirsten
University of Leipzig, Germany
{rahm, thor, aumueller, hong, golovin, tkirsten}@informatik.uni-leipzig.de

ABSTRACT

We present a new approach to information fusion of web data sources. It is based on peer-to-peer mappings between sources and utilizes correspondences between their instances. Such correspondences are already available between many sources, e.g. in the form of web links, and help combine the information about specific objects and support a high quality data fusion. Sources and mappings relate to a domain model to support a semantically focused information fusion. The iFuice architecture incorporates a mapping mediator offering both an interactive and a script-driven, workflow-like access to the sources and their mappings. The script programmer can use powerful generic operators to execute and manipulate mappings and their results. The paper motivates the new approach and outlines the architecture and its main components, in particular the domain model, source and mapping model, and the script operators and their usage.

Keywords: Data integration, Peer-to-peer system, mappings

1. INTRODUCTION

Most proposed data integration approaches rely on the notion of a global schema to provide a unified and consistent view of the underlying data sources [10]. This approach has been especially successful for data warehouses, but is also used for virtual integration of web data sources. Unfortunately, the manual effort to create such a schema and to keep it up-to-date is substantial, despite recent advances, e.g. in the area of semi-automatic schema matching [14]. Likewise, the effort to integrate new sources is usually high making it difficult to scale to many sources or to use such systems for ad-hoc (explorative) integration. Notwithstanding the high effort associated with a global schema, it cannot guarantee good data quality at the instance level. Integrating the real data, e.g. during query processing over different web sources, may still require extensive data cleaning to achieve good results, e.g. to deal with duplicate data [15].

The iFuice approach (information Fusion utilizing instance correspondences and peer mappings) focuses on the instance data of different sources and mappings between them. Many web sources expose explicit, high quality instance-level correspondences to other sources, e.g. in the form of web links. Such correspondences represent one type of mapping iFuice uses to fuse information from different sources. Sources and mappings are related to a domain model to support semantically meaningful information fusion. The iFuice architecture incorporates a mapping mediator offering both interactive and script-driven, workflow-like access to the sources and their mappings. The script programmer can use powerful generic operators to execute and manipulate mappings and their results.

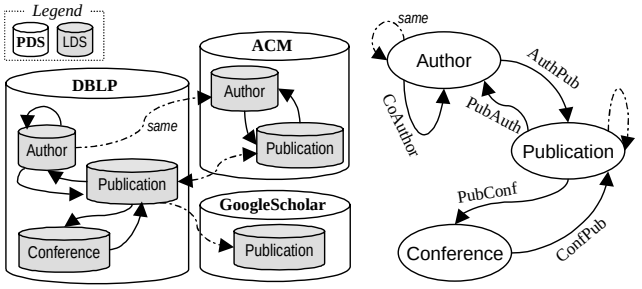
Bioinformatics is one area where this approach holds great promise. There are hundreds of web-accessible data sources on molecular-biological objects such as genes, proteins, metabolic pathways, etc. which highly cross-reference each other [5]. Creating a global schema for a sizable fraction of these sites is virtually impossible due to the high diversity, complexity and fast evolution of the data. The existing cross-references represent a low-level way to obtain additional information from other sources for a specific object, e.g. a gene. The additional information is typically of high quality since links are mostly established and maintained by domain experts. However, the manual navigation is unsuitable for evaluating large sets of objects, e.g. for gene expression analysis, so that there is a strong need for a more powerful integration approach. Moreover, the semantics of the links is typically not made explicit so that the user has to know exactly what kind of relationship they represent. The bioinformatics area also has a strong demand for experimental workflows to repetitively perform a series of analysis steps interrelating and aggregating information from different sources. The iFuice script facility aims at supporting such requirements with little development effort on the user side.

Instance-level cross-references are available in many other domains or can be generated with little effort, e.g. to interrelate bibliographic information, product descriptions and prices, etc. For simplicity we use examples from bibliographic data sources throughout this paper to illustrate the approach. Figure 1a shows three sample data sources and associated mappings. A physical data source (PDS), e.g. DBLP, may offer objects of different types. We call the object types of one PDS the logical data sources (LDS), e.g., Author, Publication and Conference as provided by DBLP. Object types combined across sources are represented in the abstract domain model (Figure 1b). Each mapping between source instances has a *mapping type* which is also represented in the domain model. Mappings map instances of an input object type to instances of an output object type, e.g. all mappings of type AuthorPubs relate author instances to their associated publications.

An important mapping type is signified by the *same-mappings* interrelating instances of the same object type across PDS, and provides a means to fuse the information for the respective instances. Typically, same-mappings are based on unique object ids, e.g. accession numbers in molecular-biological data sources or stable web URIs. Figure 1a indicates three such same-mappings, of which some already exist (e.g. DBLP links its author pages to the ACM author entries).

All mappings of the source-mapping model are executable, e.g. implemented by a query or web service. iFuice allows for explorative data fusion by browsing along these mappings, e.g. to derive from a DBLP author all publications from the directly or transitively connected LDS. The execution of several mappings and manipulation of their results can be specified within scripts to allow repeated executions for different input objects or to use the script as an implementation of a complex mapping. For example, we may want to have a script determining for a given conference

Eighth International Workshop on the Web and Databases (WebDB 2005), June 16-17, 2005, Baltimore, Maryland.



a) source mapping model b) domain model

Figure 1. Fusion scenario for a bibliographic domain (*same-mappings* are denoted by dashed lines)

X its most frequently referenced papers, e.g. to determine candidates for a 10-year best paper award. An iFuice representation of such a script is shown later. Informally, it locates conference X in DBLP, executes the PubConf mapping to get all publications of that conference, uses the same-mapping to Google Scholar to get the corresponding publications together with an attribute indicating the number of citations, sorting the publications on the number of citations, and returning the top-most publications. The example shows that mappings need to be executable on a set of input objects and return a set of output objects. Mapping execution can be restricted to specific sources or to all sources of a specific type for which a corresponding mapping implementation exists.

The main contribution of the paper is a new generic way to dynamic information fusion based on instance correspondences and executable mappings between sources. Source and mapping semantics are reflected in a domain model which is at a higher abstraction (ontological) level than a global schema and easier to construct. Mappings are executable on sets of objects and highly composable thereby supporting powerful aggregation of information over several sources. We propose different types of mappings on web data sources, including basic and aggregated mappings, same- and association mappings, and id- and query mappings. Furthermore, we introduce a set of declarative operators to execute the different kinds of mappings, to perform data aggregation (fusion), and to manipulate mapping results. Lastly, we show the usage of the operators for script programming.

In the next section, we introduce the representation of sources, mappings and the domain model, and how to add new sources and mappings. Section 3 introduces the iFuice operators on mappings and mapping results including operators for data fusion. We illustrate the use of operators by a script for the introductory example. Section 4 briefly outlines the architecture of the mapping mediator. In Section 5, we discuss related work. Finally, we conclude with a summary and outlook.

2. SOURCES AND MAPPINGS

In this section we describe the metadata used by the mapping mediator to provide uniform access to the sources and their mappings both for interactive exploration and script execution. The mediator's metadata is held in a repository and specified by a metadata model as shown in Figure 2. It consists of two main parts, a *source-mapping model* and a *domain model*. The source-mapping model describes both the accessible data sources and their associated mappings. The domain model specifies object types and mapping types. The sample models of Figure 1 conform to metadata model of Figure 2.

In the following, we describe the modeling and use of sources, mappings and the domain model. Finally we discuss the steps for adding a new source and mapping to the system. The description introduces several kinds of mappings, for which specific operators will be defined in Section 3.

2.1 Sources

We distinguish between a physical data source (PDS) and logical data source (LDS). A PDS can be a database, website, private user files or any other information base. A PDS can hold instances of different object types. We separate a PDS into LDS's each containing instances of exactly one object type of the domain model. The structure of a LDS is described by a set of attributes (Fig. 2). We only mandate the specification of an identifying key attribute per LDS to access its instances and to ensure that each instance is provided with a unique object id. Website instances are typically identified by URLs. Uniqueness for database instances can be established by concatenating instance key values with the ids of the corresponding PDS and LDS.

Requiring only an id attribute per LDS allows us to integrate a variety of heterogeneous data sources including unstructured and semi-structured data sources, e.g. websites. Furthermore, it makes it easy to add new data sources, and helps to insulate the mediator information against structural changes in the sources and thus to support a high degree of data source autonomy.

Each LDS has to provide a source-specific mapping for id-based instance access, i.e. to return for a given id the associated instance including all attribute values. We call these mappings *getInstance* mappings. The implementation of such mappings may be very simple (e.g. database lookup), but may also extract specific attributes from a webpage. The actual attributes returned are thus determined by the mapping implementation and depend on the current source content. The mapping mediator supports such variably structured result sets and their dynamic fusion with data from other sources at runtime.

In addition to the id attribute, further attributes can be optionally specified in the mapping mediator for enhanced functionality, e.g. to offer query access on the attributes or to select sources based on the availability of specific result attributes (e.g. number of citations for publications). Query access can optionally be provided by LDS-specific mappings, which we call *QueryInstances* mappings. The LDS attributes on which queries are supported should explicitly be registered in the mediator metadata.

Another optional specification is correspondences between attributes of two sources. Such correspondences are useful to enhance the fusion of instance values (see Figure 3). For instance, specifying that DBLP.Author.name corresponds to ACM.Author.author can help avoid author names appearing twice in author instances fused from DBLP and ACM. Attribute correspondences can also be used to map queries specified on one source to equivalent queries on other sources (query transformation).

2.2 Mappings

Mappings describe directed relationships between instances of two object types. They are used to uniformly interrelate instances within and between physical data sources irrespective of the underlying data management systems. The semantics of the mapping relationship is expressed by a mapping type of the domain model.

We distinguish between several kinds of mappings, including basic (simple) vs. aggregated mappings, same- vs. association mappings and id- vs. query mappings. *Basic mappings* interrelate instances of one input and one output LDS and return a result set from the output LDS. All mappings shown in Fig. 1a are such basic mappings. Note that the input and output LDS may be the

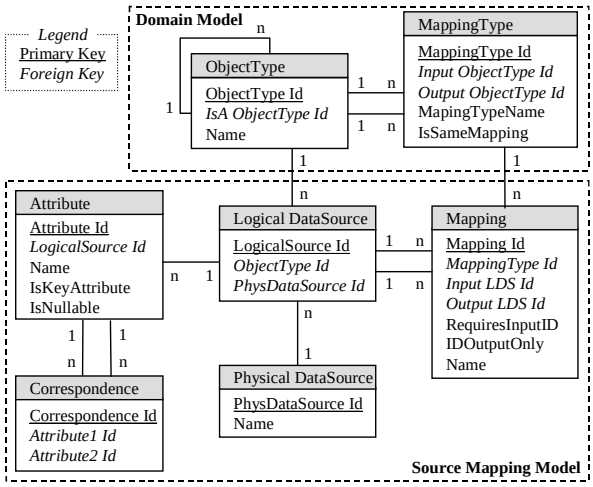


Figure 2. Metadata model of mapping mediator

same, e.g. for mappings of type CoAuthor or the source-specific getInstance and queryInstances mappings. *Aggregated mappings* are the result of mediator operations or scripts, and interrelate aggregated objects from several LDS of the same type. An example is a mapping to combine authors with their publications from several LDS (e.g., DBLP, ACM and Google Scholar).

Same-mappings represent semantic equality relationships between physical data sources at the instance level, i.e. each correspondence should refer to the same real-world object. They thus interrelate LDS of the same object type from different PDS (e.g. DBLP.Author and ACM.Author). These correspondences at the instance level are much more specific than attribute correspondences and can guide a high quality data fusion. Note that the composition of same-mappings results in new same-mappings which can thus be used to interrelate data from many sources. *Association mappings* are non-same-mappings and mostly represent domain-specific relationships between LDS of the same PDS (e.g. AuthorPubs). By composing them with same-mappings they can relate to and fuse with data of other PDS (see section 3).

Mappings can be further categorized on whether they are id-based or query-based with respect to their input instances (id- vs. query mapping). The output of basic mappings is always assumed to include the id of the returned instances. A mapping may in fact only return ids, e.g. as input for subsequent id-mappings and to limit the amount of data to transfer to the mediator and to process there.

Id-mappings interrelate ids (and thus instances) of two LDS or PDS and can easily be composed. The result of id-mappings can be represented by a set of instance correspondences (id1, id2). *Query mappings* are helpful to find relevant instances and their ids in the first place. One example are source-specific QueryInstance mappings. Their results include instance ids and can thus be combined with id-mappings to obtain related query results from different sources. The attribute values of a query result may also be used to query another source.

The mapping specification in Figure 2 only considers basic mappings for simplicity. It derives the distinction between same- and association mapping from the used mapping type of the domain model. The attribute *RequiresInputID* indicates whether or not the mapping is id-based.

The implementation of mappings can use other mappings, utilize database queries, etc. To hide implementation differences, iFuice mappings are uniformly encapsulated as web service operations and use XML for data exchange. Ideally, id-mappings, including same-mappings, can be based on existing instance correspondences such as web links. Alternatively, they may be implemented by a query mapping, e.g. to use instance values from one source (e.g. obtained from a getInstance mapping) to search for corresponding instances of a second source (input for query mapping). For instance, the same-mapping between DBLP publications and Google Scholar can be implemented by using the name and author of a DBLP publication as a keyword query to Google Scholar.

For improved performance, the results of id-mappings, i.e. the set of instance correspondences, may be stored or cached in binary (id/id) mapping tables [8]. Composition between such mappings then becomes a join operation. Materialized id-mappings can also be inverted, even for n:m cardinalities (e.g., an AuthorPub mapping can be derived from an Id-based PubAuthor mapping and vice versa).

2.3 Domain model

The domain model defines domain-specific object types and mapping types to semantically (ontologically) categorize data sources and mappings. A hierarchical (taxonomical) categorization of object types is possible to classify sources in more detail (e.g., conferences based on discipline). We do not include attributes for object types to accommodate a large variety of data sources and to make it much easier to construct the domain model than a global schema. In many cases, we expect a small set of object types to be sufficient. New object types may be added as needed to accommodate new sources, i.e. the domain model can be incrementally extended in a bottom-up fashion. A mapping type interrelates two object types. A special attribute indicates whether the mapping type represents same-mappings (semantic equality relationship).

2.4 Adding Sources and Mappings

One goal of iFuice is to make it easy to add new sources and mappings. A new physical data source requires to register at least one logical data source. Registering a LDS requires to assign it to the corresponding object type, specification of an id attribute, and provision of a getInstance mapping. Furthermore, a peer mapping P to at least one other LDS should be provided to permit data fusion with other sources. Optionally, a QueryInstances mapping can be provided, and additional attributes (e.g. known output attributes of P or for query input) and attribute correspondences can be specified. For a LDS of a new object type, the domain model must be extended with the corresponding object type (e.g. Journal) and at least one associated mapping type (e.g. JournalPubs).

Provision of a new mapping requires its registration at the mapping mediator. This involves the specification of the mapping characteristics and possibly the registration of a new mapping type in the domain model. The mapping must be executable, i.e. an implementation must be provided. The mapping implementation can hide many details of the underlying data sources and typically exposes only selected input and output attributes at the interface. As discussed, we do not require that the input and output attributes of a mapping be registered in the source-mapping-model.

To illustrate the ease and benefit of providing data sources and mappings consider the following simple example: A user keeps a list of her favorite authors (including handpicked information like e-mail address or nationality, which are accessible by a *getInstance* operation) in a local file and wants to bind it to the mapping mediator so that she can periodically check the information about the authors' publications. This can be achieved by establishing a same-mapping between the local file and the LDS *DBLP author*, e.g. by providing a list of DBLP URLs. Thereafter the ex-

isting mappings between DBLP, ACM and Google Scholar can be used to gather the information of interest.

3. OPERATORS AND SCRIPTS

Interactive users and script programmers should not be limited to the execution of one mapping at a time but are provided with more powerful operators including data fusion capabilities. Therefore, the iFuice mapping mediator supports a variety of operators which can be used within script programs or to implement derived mappings. This idea is inspired by the script approach for model management, e.g. as implemented in Rondo [11]. While Rondo focuses on metadata manipulation, iFuice provides operators for mapping execution and manipulation of instances.

We designed operators of different complexity to allow users to focus access on specific data sources and mapping paths. Compared to transparent access on many sources, this not only helps improve performance but is also important for user acceptance and data quality. This is because users often have specific preferences for some data sources, and the cleanliness of merged data tends to decrease with more sources. Therefore we designed two sets of operators, one for processing basic mappings returning objects from one source (getInstances, traverse, map, queryInstances, queryTraverse queryMatch) and one for aggregated mappings and aggregated objects (aggregateSame, aggregateQueryTraverse aggregateMap, fuseAttributes). In both cases we have a set of operators to process the respective results, similar to query languages (union, intersect, project, sort, join, ...). All operators are set-oriented, i.e. they work on sets of simple or aggregated input objects and determine a set of result objects or a mapping.

The next two subsections introduce these two sets of operators. In 3.4 we present a script program using the operators.

3.1 Operators for basic mappings

Basic mappings relate instances of one input LDS with instances of one output LDS, i.e. we obtain a homogeneous set of result objects. An *object* (instance) o_i consists of a unique id plus a (possibly empty) list of attribute values. The id is assumed to also identify the logical data source to which the object belongs. We first introduce operators for id-mappings and then for query mappings.

3.1.1 Operators for id-mappings

Let L_1, L_2, … denote logical data sources, O_1, O_2, … sets of L_1 objects, L_2 objects … and m_1, m_2, … id-mappings (same- or association mappings).

traverse $(O_1, m_2, \ldots, m_k) \rightarrow O_k$

traverse $(O_1, m_2, \ldots, m_k) = m_k(m_{k-1}(\ldots m_2(O_1)))$

consecutively executes m_2, m_3 … thereby traversing via $L_2 - L_3$ to L_k. Of course, the input source of m_i must correspond to the output source of m_{i-1}. Note that both same- and association mappings can be used within a traversal path. For same-mappings the LDS names can be used instead of the mapping names. The output is required to be a set, i.e. no duplicates are allowed. *Example*: Let O_1 be a list of DBLP author URLs, traverse (O_1, ACM, ACMAuthorPub) returns their corresponding ACM publications. traverse (O_1, DBLPAuthPub, DBLPPubConf) returns the conferences in which the authors in O_1 have published (without returning the authors and publications).

For same-mappings we provide a variation of traverse

traverseSame $(O_1, LDS_k) \rightarrow O_k$

It is not restricted to a single traversal path but considers all paths of same-mappings from the input LDS_1 to LDS_k and takes the union of their LDS_k results.

The traverse and traverseSame operators only return the instances of the last LDS on the mapping path which can thus be the input for other operators on objects. Frequently one wants to see the correlations between objects of the first and last source. This is achieved by

map$(O_1, m_2, \ldots, m_k) \rightarrow O_1 \times O_k$

map$(O_1, m_2, \ldots, m_k) = \{(o_1,o_k) | o_1 \in O_1, o_k \in \text{traverse}(\{o_1\}, m_2, \ldots, m_k)\}$

In the special case k=2, map returns the instance correspondences of a single mapping (there may be just id-id combinations, e.g. for same-mappings). For more than one mapping, the semantics corresponds to that of a classical compose operation. *Example*: map (O_1, DBLPAuthPub, DBLPPubConf) returns authors together with the conferences in which they published, i.e. these different instances are 'fused' together by the mapping result.

The support operator

getInstances $(O_1) \rightarrow O_1$

determines for the input instances in O_1 the available attribute values. This operator usually is applied to objects that only hold an id value or a subset of attributes. *Example*: Given a list of DBLP author URLs, getInstances adds attribute values, e.g. name, no. of co-authors etc. In the previous operators, the implementation of the mappings, especially m_k, determines which attributes are present in the output instances. Applying the getInstances operator on these instances helps to obtain additional attribute values if needed.

3.1.2 Operators for (basic) query mappings

In iFuice, queries are posed for one source and can then be propagated to other sources by applying id-mappings or query matching. For querying a source we use

queryInstances : $(L_1, \{cond\}) \rightarrow O_1$

which returns all object instances (i.e. at least their ids) from L_1 which fulfill the given set of attribute conditions $\{cond\}$.

To propagate a query, we use derived operators combining queryInstances with traverse or traverseSame.

queryTraverse $(L_1, \{cond\}, m_2, \ldots, m_k)$
= traverse (queryInstances $(L_1, \{cond\}), m_2, \ldots, m_k) \rightarrow O_k$

queryTraverseSame $(L_1, \{cond\}, L_k)$
= traverseSame (queryInstances $(L_1, \{cond\}), L_k)$

Example: queryTraverseSame (DBLP, {name= 'Bernstein'}, ACM) returns the ACM author objects of all DBLP authors with that name.

Operator queryMatch transforms an input query for one source to an equivalent one on a second source:

queryMatch $(L_1, \{cond\}, L_2)$ = queryInstances $(L_2,$ attrTransf $(\{cond\}))$

The function attrTransf utilizes specified attribute correspondences to map the L_1 query condition into a corresponding L_2 query condition. Hence, queryMatch is only applicable to L_2 sources for which a queryInstances implementation and attribute correspondences have been provided. This operator typically is used to build the union of the source-specific results which leads to aggregated objects.

3.2 Operators for aggregated mappings

Same-mappings identify semantically equivalent objects (synonyms, duplicates) which should be combined to reduce redundancy and merge (complement) the available information from different sources. We separate this into two steps, called aggregation and fusion. Figure 3 exemplifies this for two semantically equivalent publication objects. In step 1 we combine them into one aggregated object which is a combination of all attributes from the original objects. In step 2 we fuse the attributes to reduce

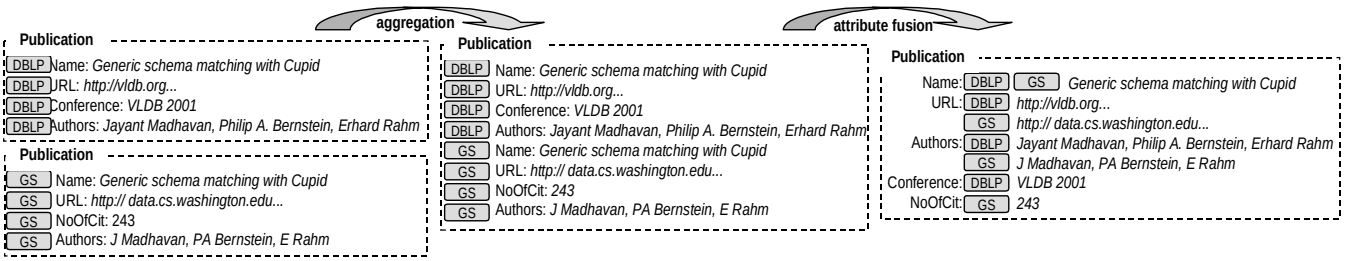


Figure 3. Example for aggregation and attribute fusion

redundancy without losing information. The first step is usually the most difficult one, but is well supported in iFuice by the same-mappings thereby facilitating good data quality. The second step can use attribute correspondences or actually analyse the existing values for merge possibilities.

Most iFuice operators for aggregated mappings deal with aggregated objects where all attribute values are still available for further processing. The fusion of attribute values is performed by a separate operator, fuseAttributes, which is best applied before returning aggregated objects to the user.

Let $\{o_1, o_2, o_3, \ldots\}$) be a set of semantically equivalent objects of object type T from one or more logical data sources. The aggregation of these objects agg $(\{o_1,o_2,o_3,\ldots\}) = (o_1\text{-}o_2\text{-}o_3\text{-}\ldots)$ is called an *aggregated object* and also refers to object type T.

disagg $(o_1\text{-}o_2\text{-}o_3\text{-}\ldots) = \{o_1,o_2,o_3,\ldots\}$ returns the components of an aggregated object.

Aggregating objects of the same type

Let AO_1, AO_2, … be sets of aggregated objects of type T, and L_2 a LDS of object type T. The operator

aggregateSame $(AO_1, L_2) \rightarrow AO_2$

$=\{\text{agg}(ao_1,\{\text{traverseSame}(\{o_1\},L_2) \mid o_1 \in \text{disagg}(ao_1)\}) \mid ao_1 \in AO_1\}$

aggregates all AO_1 objects with semantically equivalent objects in L_2 by evaluating the same-mappings from the input objects to the corresponding L_2 objects. Note that the operator can also be applied to simple input objects from one input LDS. Note further that the same-mappings implement the duplicate detection and can thus support efficient and high quality data aggregation. The definition of aggregateSame can easily be generalized to more than one target LDS since the operator works on sets of aggregated objects, e.g. the output of a previous aggregateSame execution.

Combining the results for a propagated query at different sources leads to aggregated objects and is supported by

aggregateQueryTraverse $(L_1,\{cond\}, L_k)$

= aggregateSame (queryInstances $(L_1,\{cond\})$, L_k).

The standard set-oriented (relational) operators intersect, diff, union, restrict, project, sort etc. can be extended to deal with aggregated objects and duplicates. Due to space constraints we only define intersection.

intersect $(AO_1, AO_2) = \{(ao_1\text{-}ao_2) | ao_1 \in AO_1, ao_2 \in AO_2, ao_1 \approx ao_2\}$

Thereby, $\approx$ denotes that two aggregated objects are semantically equivalent. This is the case if they share at least one component object or if they are related by a same-mapping.

Aggregating objects of different types

Association mappings typically interrelate objects of different types which should not be aggregated together like equivalent objects. For these mappings, we generalize the traverse operation to both mapping types and aggregated objects.

aggregateTraverse $(AO_1, mt) \rightarrow AO_2$

$=\{\text{agg}(\{\text{traverse}(\{o_k\},m) | o_k \in \text{disagg}(ao_k), m \text{ of } mt\}) | ao_k \in AO_1\}$

applies all association mappings of type *mt* for all objects in AO_1 and aggregates the resulting objects. Similarly, we generalize the map operator for association mappings and aggregated objects to obtain binary aggregated mappings:

aggregateMap $(AO_1, mt) \rightarrow AO_1 \times AO_2$

$=\{(ao_1, ao_2) \mid ao_1 \in AO_1,\ ao_2 \in \text{aggregateTraverse}(\{ao_1\}, mt)\}$

Example: Given a set AO_1 of aggregated objects of DBLP and ACM authors, aggregateMap (AO_1, AuthPubs) returns author-publication pairs of aggregated objects that contain object instances from DBLP or ACM or both.

For aggregated mapping results $MR_1 \subseteq AO_1 \times AO_2$ and $MR_2 \subseteq AO_3 \times AO_4$ the operators join and compose are defined as follows

join (MR_1, MR_2)

$= \{(ao_1,(ao_2\text{-}ao_3),ao_4) | (ao_1,ao_2) \in MR_1, (ao_3,ao_4) \in MR_2, ao_2 \approx ao_3\}$

compose$(MR_1,MR_2) = (\{ao_1,ao_4\} | (ao_1,ao_2,ao_4) \in$ join $(MR_1, MR_2))$

The given join semantics refers to an inner join, but left outer join etc. can be specified analogously. Join and compose also need a duplicate detection to aggregate semantically equivalent objects.

3.3 Script Example

A script is a sequence of operator calls. Each operator call stores the results into a variable (denoted by a '$'-prefix). The following simple script example presents an approach to determine candidates for the 10-Year Best Paper Award.

```
$SIGMODPubs := queryTraverse (LDS=DBLP.Conf, {Name="SIGMOD 1995"},
                                                DBLPConfPubs)
$CombinedConfPub:= aggregateSame ($SIGMODPubs, GoogleScholar)
$CleanedPubs := fuseAttributes ($CombinedConfPub)
$Result := sort ($CleanedPubs, "NoOfCitings")
```

In this example, step 1 uses a queryTraverse operation to query on the LDS DBLP.Conf to determine the DBLP id for the conference of interest and traversing to the associated publications. The used mapping DBLPConfPubs is assumed to determine complete instances. Step 2 utilizes the same-mapping on publications between DBLP and Google Scholar to aggregate the DBLP values with the corresponding instances in Google Scholar. In step 3 we clean the aggregated objects by applying fuseAttributes. Finally, we sort the resulting set of fused publications on the attribute denoting the number of citations. The top items/publications in the final result set indicate likely candidates for the 10-Year Best Paper Award.

Since operators can process many input objects at a time, the script does not only apply to a single conference but many. To determine the most-cited publications of, say, a whole conference series, one could use a modified query in step 1 to select these conferences, e.g. SIGMOD or VLDB. The rest of the script can remain unchanged.

4. MEDIATOR ARCHITECTURE

Figure 4 gives an overview of the iFuice mediator architecture. Its main components are the *repository* (already described in Section

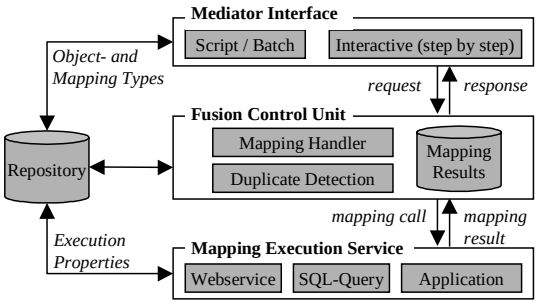


Figure 4. Architecture of the iFuice mediator

2), *mediator interface (MI)*, the *fusion control unit (FCU)*, and the *mapping execution service (MES)*. The MI provides two modes of operation. The interactive mode supports an explorative approach where users can execute mappings step by step. The script-based mode defines a batch of execution steps for the mediator, returning a final result set. Operations are executed within the FCU, the central unit of the system. The mapping handler coordinates single mapping calls according to the appropriate mapping definition of the metadata model. It manages (temporary) mapping results for further operations. It also performs duplicate handling for aggregating objects. The mapping execution service actually executes the called mappings and returns their results to the FCU.

As a proof of concept we have implemented a first subset of the mediator functionality for interactive mapping execution (explorative navigation). The prototype is implemented in Java and utilizes a relational database system for the repository and mapping results. It can execute mappings and operation sequences as in the examples shown, including the example on the 10-year best paper award.

5. RELATED WORK

Most previous data integration approaches for web data sources utilize a global schema and a query mediator. In contrast to iFuice, these approaches do not utilize peer mappings and instance correspondences. Moreover, the global schema tends to be much more complex than our domain model leading to increased effort to add sources or to deal with changing sources.

More related to our approach is recent work on P2P databases and biological databases. P2P prototypes such as PeerDB and Piazza [12][17] focus on query processing across peer mappings without a global schema. PeerDB propagates IR searches, whereas Piazza reformulates queries based on metadata mappings. Queries refer to the peer schema where the query was initially posed. By contrast, iFuice focuses on instance-level correspondences and can apply a variety of executable peer mappings including queries. Moreover, we support a set of powerful operators (including aggregation operators) and the execution of script programs.

Several integration approaches in the bioinformatics area [16], [9], [7] utilize cross-references at the instance level to combine data from different sources. Systems like SRS [4] and our GenMapper prototype [1] materialize instance correspondences in mapping tables for improved performance. These efforts lack a sufficient consideration of the semantics of the cross references but expect the users to know what the cross references mean. The iFuice domain model differentiating different mapping types, including different same-mappings, allows a much more focused data fusion. To our knowledge, the proposed framework of mapping operators and scripts is also unknown so far in the bioinformatics domain.

SEMEX [3] is an interesting personal information management system which utilizes a domain model and mappings similar to our approach. However, it only deals with centrally stored data, while we integrate data from different web data. Most previous work on data cleaning was done in the data warehouse area with a focus on duplicate identification [1][15]. The use of semantic object-ids for data integration in the TSIMMIS mediator [13] has similarities to the utilization of ids in our *same-mappings*.

6. CONCLUSION AND OUTLOOK

iFuice combines a set of techniques to a new approach for integrating information from diverse web data sources. It does not depend on a global schema and utilizes explicit instance correspondences and executable peer mappings. We proposed the use of a domain model and a mapping mediator to control the execution of a variety of such mappings. Furthermore, we introduced a set of powerful operators for mapping execution and data aggregation. An initial prototype for interactive mapping execution showed the viability and flexibility of the approach.

In future work, we will fully implement the outlined approach and investigate techniques based on caching mapping tables to improve performance. We plan to adopt the iFuice implementation to different domains and to support integration of both web data sources and local / private data sources.

Acknowledgements. A. Thor and N. Golovin are funded by the German Research Foundation (DFG) within the Graduiertenkolleg "Knowledge Representation". H. Do and T. Kirsten are supported by DFG grant BIZ 6/1-1. Phil Bernstein and Sergey Melnik gave helpful comments.

7. REFERENCES

[1] Chaudhuri, S. et al.: *Robust and efficient fuzzy match for online data cleaning*. Proc. SIGMOD 2003

[2] Do, H.-H., Rahm, E.: *Flexible integration of molecular-biological annotation data: The GenMapper Approach*. Proc. of EDBT 2004

[3] Dong, X., Halevy, A. Y.: *A platform for personal information management and integration.* CIDR 2005

[4] Etzold, T. et al.: *SRS: An integration platform for databanks and analysis tools in bioinformatics*. In [9]: 109-145.

[5] Galperin, M.Y.: *The molecular biology database collection - 2004 update*. Nucleic Acids Research 32, Database issue, 2004.

[6] Greco, S. et al: *Integrating and managing conflicting data*. Proc. Of Conf. on Perspectives of System Informatics. 2001

[7] Hernandez, T, Kambhampati, S: *Integration of biological sources: current systems and challenges ahead*. SIGMOD Record 33(3), 2004

[8] Kementsietsidis, A. et al.: *Mapping data in peer-to-peer systems:semantics and algorithmic issues*. Proc. SIGMOD 2003

[9] Lacroix, Z., Critchlow T. (Eds.): *Bioinformatics: Managing Scientific Data.* Morgan Kaufmann, 2003

[10] Lenzerini, M: *Data integration: a theoretical perspective*. Proc. PODS 2002

[11] Melnik, S. et al.: *Developing metadata-intensive applications with Rondo*. Journal on Web Semantics, 2003

[12] Ng, W. S., *et al.*: *PeerDB: A P2P-based System for Distributed Data Sharing.* Proc. ICDE 2003

[13] Papakonstantinou, Y. et al.: *Object Fusion in Mediator Systems*. Proc. VLDB 1996

[14] Rahm, E., Bernstein, P. A.: *A survey of approaches to automatic schema matching*. VLDB Journal, 10(4), 2001

[15] Rahm, E., Do, H.-H.: *Data cleaning: problems and curren approaches.* IEEE Bull. Techn.Com. Data Engineering, 23 (4), 2000

[16] Stein, L. D.: *Integrating biological databases.* In Nature Review Genetics, 4, 2003

[17] Tatarinov, I., *et al.*: *The Piazza peer data management project.* SIGMOD Record, 32(3), 2003