On semi-automated matching and integration of database schemas

Ünal Karakaş, Ö.

Publication date
2010

Chapter 5

SASMINT development architecture

We have implemented the SASMINT system to verify, test, and validate the approach proposed in this research as well as to compare it with other approaches. Details of the development architecture of SASMINT are provided in the following sections, together with a number of screenshots of this system. Section 5.1 goes over the processing steps of SASMINT. Section 5.2 briefly addresses the technologies applied in the development of SASMINT. Section 5.3 lists the main components of the system. Detailed explanations about the operation of the SASMINT system are provided in Section 5.4. Finally, Section 5.5 concludes this chapter.

A part of the research results presented in this chapter was previously published in two articles, of which one appeared in the Journal of Knowledge and Information Systems (Unal & Afsarmanesh, 2010), and the other appeared in the Journal of Software (Unal & Afsarmanesh, 2009).

5.1 Processing Steps of SASMINT

SASMINT realizes the approach described in Chapter 4, following the processing steps shown in Figure 5.1. To state it briefly, the configuration step allows the user to first identify the weights for matching algorithms and then identify the strategy for selection of the results of schema matching. Schema matching step starts with the preparation sub-step that translates both recipient and donor schemas into the DAG format. Then, based on the approaches explained in Chapter 4, in the comparison sub-step, the linguistic and structure matching take place. After the matching results are generated by SASMINT, users apply their desired modifications and/or validate the proposed matches before the eventual match results are formally defined and persisted. Using these results, the SASMINT system then starts generating an integrated schema, by applying a set of integration rules defined for relational databases. When the results of schema integration are ready, these are again presented to the user for modification/validation of their integration.

5.2 Technologies Applied

Considering the practical environment of collaborative networks that will need to run SASMINT, this system is implemented in Java programming language on Microsoft Windows operating system environment. Being java-based, SASMINT can actually run on multiple operating platforms/systems; i.e. it runs cross-platform. We have therefore also tested it for

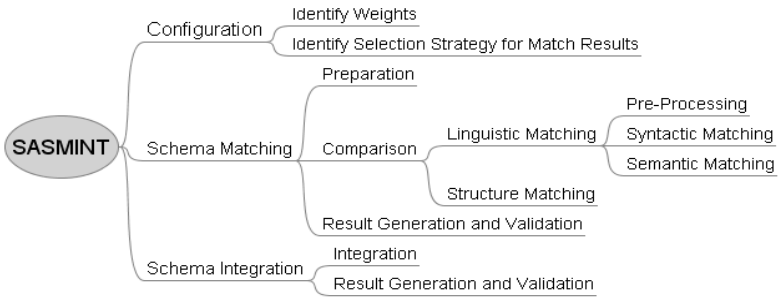

Fig. 5.1. Processing Steps of SASMINT

this purpose and within hours, this system was up and running on MacOSX platform, which actually is a Unix clone from the Operating System point of view. SASMINT is a standalone java swing based application. As far as its architecture is concerned, it has a 2-tiered architecture, where its 'business' tier provides services/functions to its presentation tier. These business layers tiers encapsulate implementations of several business rules.

We have focused our development efforts on supporting the matching and integration of relational schema based systems. The specific technologies and tools exploited in the development of SASMINT are listed below.

- *Eclipse:* Java Integrated Development Environment (IDE). It is a platform for building integrated web and application development tooling (Eclipse, 2010).
- *NetBeans*: Java Integrated Development Environment (IDE). It is a platform for building integrated java applications. We have extensively used NetBeans for the GUI design phases.
- *AWT (Abstract Windowing Toolkit) and Swing:* Java's Graphical User Interface (GUI) libraries.
- *JGraphT:* Free Java graph library to create (model) graphs (Jgrapht, 2010). It supports various types of graphs, such as weighted, unweighted, directed, undirected, and labeled graphs.
- *JGraph:* Graph component for visualization and layout (Jgraph, 2010). Graphs generated using JGraphT can be visualized and the layout can be applied by means of JGraph.
- *WordNet:* A lexical dictionary (Fellbaum, 1998; Wordnet). Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by a number of semantic relationships, such as hypernymy and hyponymy.
- *JWNL (Java WordNet Library)*: A Java API for accessing WordNet (Jwnl, 2010).
- *XML-Beans:* Technology for accessing XML by binding it to Java types (Xmlbeans, 2010). XMLBeans uses XML Schema to compile Java interfaces and classes that can be used to access and modify XML instance data.
- *SecondString:* An open source Java package consisting of implementation of a number of string similarity metrics (Secondstring, 2010). For the purpose of

syntactic matching in SASMINT we modified this package for implementing the Levenshtein, Monge-Elkan, Jaro, TF*IDF, and Jaccard metrics.

5.3 Main Components of the System

The main components of the SASMINT system are illustrated in Figure 5.2. The *Sampler Component* helps users identify appropriate weight for each algorithm used in the linguistic matching. The *Graph Representation Component* of SASMINT is responsible for representing schemas in Graph format, more specifically in the DAG format. SASMINT uses JGraph for graphical representation (visualization) and specifying the layout of the graphs. This system utilizes the Java graph libraries of JGraphT, for creating a graph and performing some operations on the graph, such as getting all its vertices and edges and traversing the graph. Users interact with the system using the *GUI Component*. After the schemas, represented as graphs, have been displayed using the GUI and the user has selected the Match option, the GUI component calls the *Schema Matching Component.* This component matches source and target schemas using a combination of Linguistic and Structure Matching techniques, as explained in Chapter 4. After modifying and validating the match results, the user may continue with schema integration. The *Schema Integration Component* integrates the schemas using a number of pre-defined rules and represents the automatically generated integrated schema to the user, while also formalizing it in a derivation language. Results of integration will go again through the user validation stage.

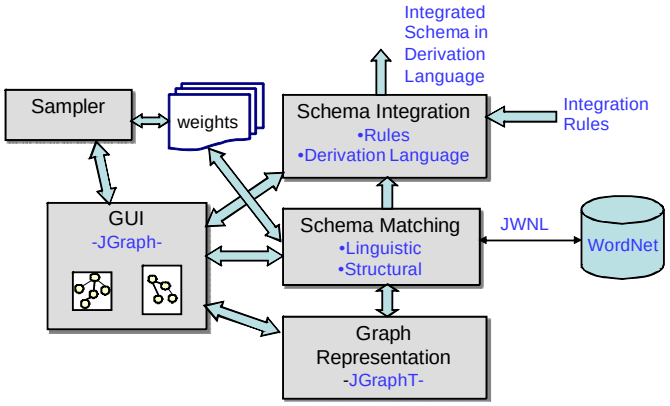

Fig. 5.2. Components of SASMINT

5.4 How does the System Work?

The main flow of information in the SASMINT system as well as the technologies used in different stages is given in Figure 5.3. Firstly, weight for each schema matching algorithm is assigned and the strategy for selecting the match results is identified and threshold value is set, as explained in details in Section 5.4.1. Secondly, the two schemas, called as recipient and

donor, are translated into the graph format and then loaded into the system to be visualized by the SASMINT GUI. Thirdly, correspondences between these two schemas are identified. Finally, the match results are presented to the user, and after modifying and/or validating the match results, if the user continues with the schema integration according to specified matches, these two schemas are integrated into a single schema and then it will be formalized in the SDML, which again requires user validation.

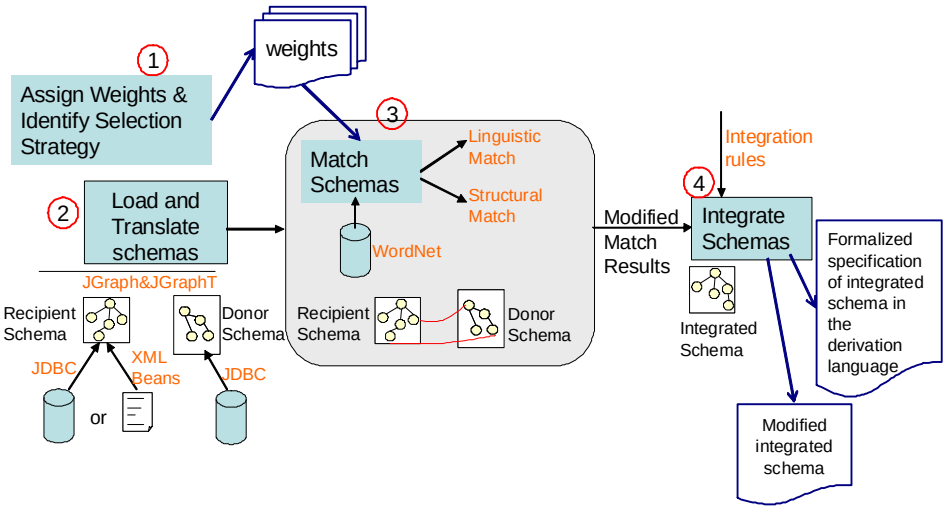

Fig. 5.3. Information Flow in SASMINT

In the following sections, some more details of the operation of the SASMINT system are provided. For this purpose example schemas shown in Figures 5.4-a and 5.4-b are used. Through the screenshots, also the matching and integration of recipient and donor schemas are explained.

```
CREATE TABLE `employee` (
  `empid` int(10) unsigned NOT NULL
auto_increment,
  `fname` varchar(45) NOT NULL,
  `lname` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
  PRIMARY KEY (`empid`)
)
```

Fig. 5.4-a. Recipient Schema

```
CREATE TABLE `address` (
  `addrid` int(10) unsigned NOT NULL auto_increment,
  `street` varchar(45) NOT NULL,
  `zip` varchar(45) NOT NULL,
  `city` varchar(45) NOT NULL,
  PRIMARY KEY (`addrid`)
)
CREATE TABLE `person` (
  `pid` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `birth` varchar(45) NOT NULL,
  `addressid` int(10) unsigned NOT NULL,
  PRIMARY KEY (`pid`),
  KEY `FK_person_1` (`addressid`),
  CONSTRAINT `FK_person_1` FOREIGN KEY (`addressid`) REFERENCES
`address` (`addrid`)
)
```

Fig. 5.4-b. Donor Schema

5.4.1 Assigning Weights and Identifying the Selection Strategy

In the three methods introduced for the configuration phase of SASMINT in Chapter 4 (Section 4.2.2), it is stated that in one method, the weights for each algorithm used for schema matching can be manually assigned, as shown in Figure 5.5, while in another method, default weights are assigned by the SASMINT system. The default weight for each algorithm is the equal weight in the related category of matching. For example, the default weight for the six considered metrics in the syntactic similarity is 1.0 / 6, which is ≈0.16. Furthermore, for combining the results of linguistic and structure matching for the final result, default weight of linguistic matching is 0.7 and that of structure matching is 0.3. As explained in Section 4.2.3.3, these weights are also modifiable through the GUI, shown in Figure 5.5.

Since it may be difficult for an average user to identify appropriate weights, SASMINT introduces and supports a third method implemented by the Sampler component for automatic weight identification, indicating which algorithm/algorithms better suited for the specific domain of the schemas. Sampler can be applied to the calculation of both syntactic and semantic similarity metrics.

With the current implementation, the Sampler component can work with up to five known sample pairs (this is merely a design feature, and is thus easily customizable). Through the GUI provided by the Sampler component, as shown in Figure 5.6, the user has the freedom to provide: a) syntactically similar pairs in case he/she would like the system to compute the weights of syntactic matching metrics, or b) semantically similar pairs in case he/she requires to compute the weights of metrics for semantic matching. The user is expected to input these pairs to the Sampler component from his/her schema domain. For instance, the user might want to see how syntactic similarity metrics would perform for the pair P: ["student_name", "name_of_student"]. On the other hand, he might want to see how semantic similarity metrics would perform for the pair P: [“employee”, “worker”]. After doing some computations, as explained in Chapter 4, the Sampler component outputs the appropriate weights for the schemas’ domain, as can be seen in Figure 5.6.

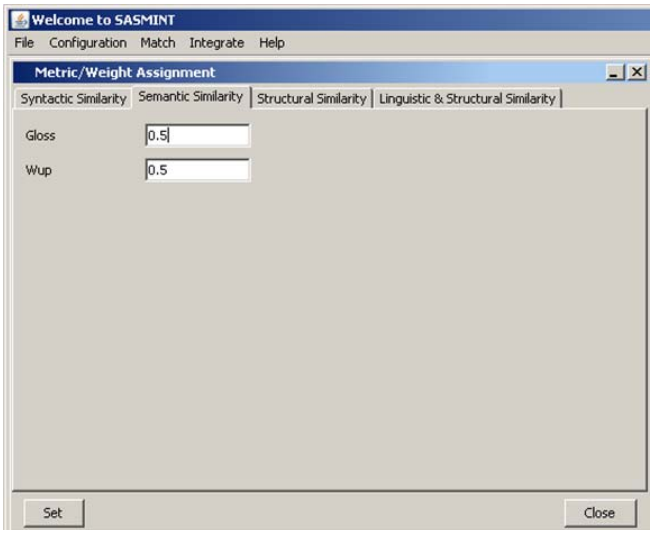

Fig. 5.5. Manual Weight Assignment

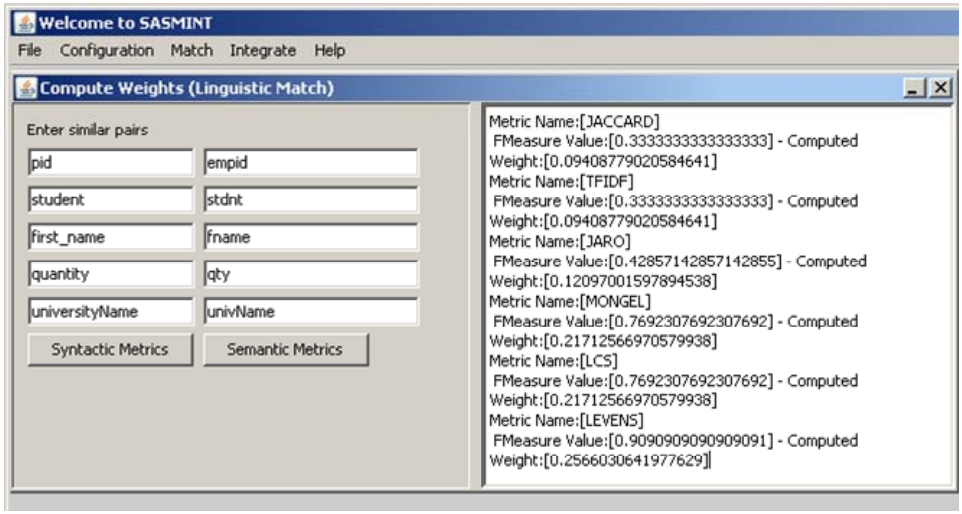

Fig. 5.6. Use of Sampler

For identifying the selection criteria for the results of schema matching, user is required to set a threshold value as well as the strategy to use for selecting the results of schema matching. Figure 5.7 shows the screenshot relevant to this example from SASMINT. The meanings of 'select all above threshold' and 'select max above threshold' are explained in Chapter 4 (Section 4.2.2). If nothing is specified by the user, default value for threshold is 0.5 and default strategy is "select max above threshold".

Fig. 5.7. Identifying Result Selection Strategy

5.4.2 Loading and Translating Schemas

After assigning weights to the algorithms and identifying the selection strategy (e.g. 'select all above threshold' and 'select max above threshold'), two schemas, called the *recipient* and *donor* respectively, are loaded by the user into SASMINT, through its GUI, as shown in Figure 5.8. The recipient schema is taken as the base schema in the integration process. It can be loaded either from a database or from an XML file, which contains a previously generated integrated schema and all its related derivation information. Format of this file is described in the following paragraphs about the SDML. It is assumed that XML file contains an SDML-based representation of the integrated schema, which is loaded as the recipient schema. On the other hand, donor schema can be loaded from any database. During the loading process, schemas specified in the language of their database are translated into a Directed Acyclic Graph (DAG) format. In other words, table and column names as well as the primary and foreign keys will all be represented in graphs for the recipient and donor schemas, before their processing starts. When a recipient schema is loaded from an XML file, only this information is shown in the graph, not its derivation information. DAG is used as the common format for representing schemas. SASMINT uses JGraphT, a free Java graph library (Jgrapht, 2010), to create the DAG.

When the two graphs, corresponding to donor and recipient schemas are generated, they are displayed for user through the SASMINT GUI. A graph component, called JGraph and its subcomponent, JGraph Layout are used for graph visualization and layout (Jgraph, 2010). The processes of SASMINT, responsible for loading schemas and translating them into the graph format constitute the *Preparation* sub-step.

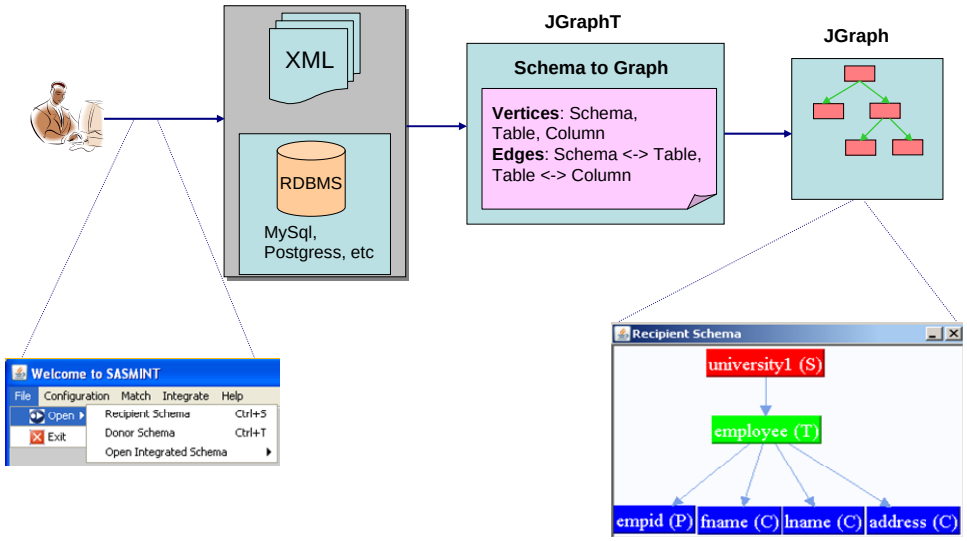

Fig. 5.8. Preparation Sub-step

Figure 5.9 illustrates the screenshots of the two schemas given in Figure 5.4 in the MySQL database. Figure 5.10 shows how they are represented as graphs when loaded into SASMINT. Schemas, tables, and columns are shown in different colors to help users during the modification and validation of the schema matching results. Furthermore, primary and foreign key columns are also indicated in the views, by appending a "(P)" for a primary key and a "(F)" for a foreign key in the node name. JGraph provides a flexible visualization for graphs. It allows users to easily move the nodes in order to modify the appearance of a graph.

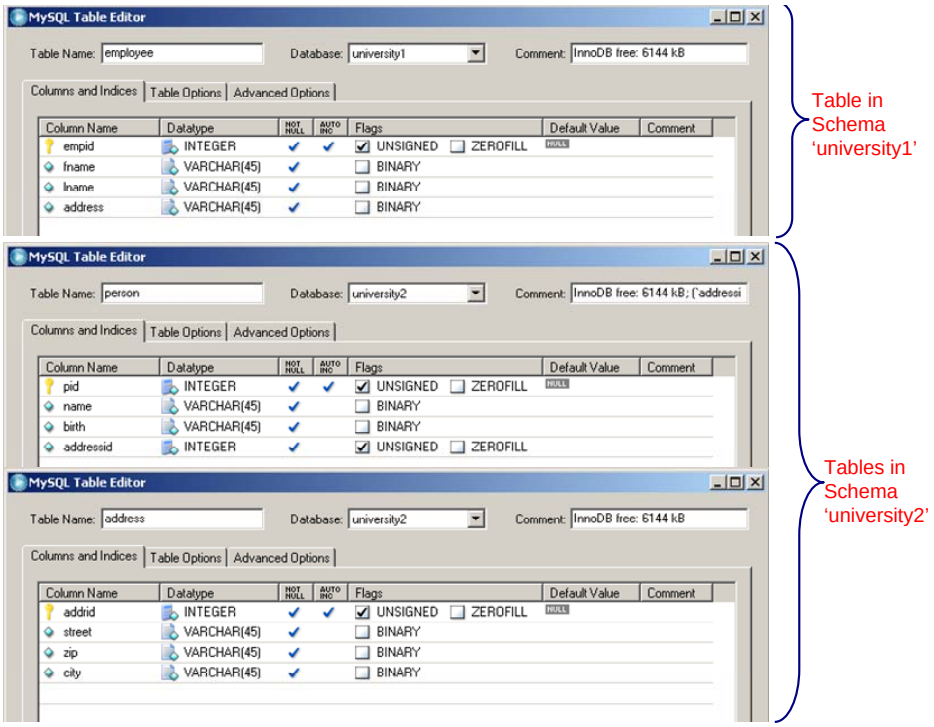

Fig. 5.9. Recipient and Donor Schemas in the Database

5.4.3 Matching Schemas

After displaying recipient and donor schemas, user selects the Match option from the menu. This option computes the similarities between the elements of these two schemas using the matching algorithms explained in Chapter 4. Results are filtered first based on the threshold value. As explained in Section 5.3.1 and shown in Figure 5.7, the value for threshold is set by the user before schema matching starts or default value is used. This value is used by schema matching for selecting the pairs, for which their similarities are above the threshold, and then the second filtering is applied using the selection strategy.

After applying these two filters, remaining similar pairs are displayed to the user for modification and validation. For the example university schemas, a screenshot of the system after schema matching is shown in Figure 5.11. In addition to a graph with "Similar To" edges

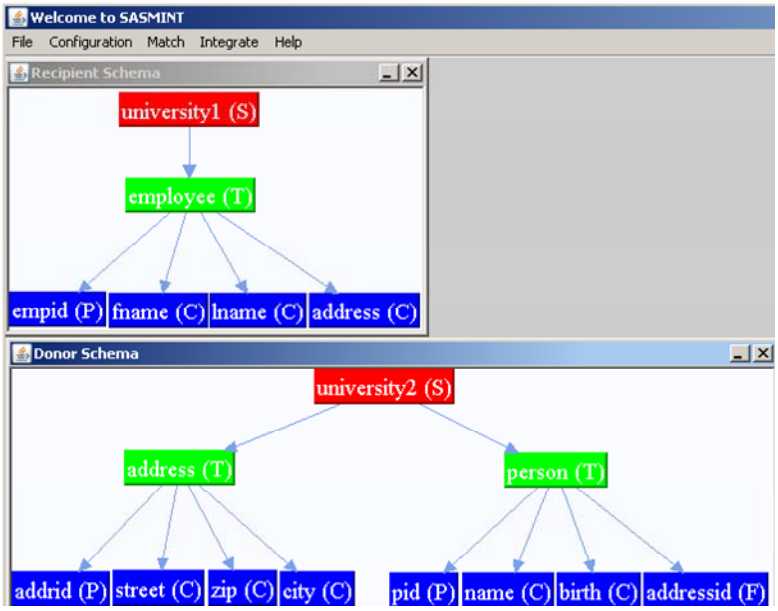

Fig. 5.10. Recipient and Donor Schemas Loaded

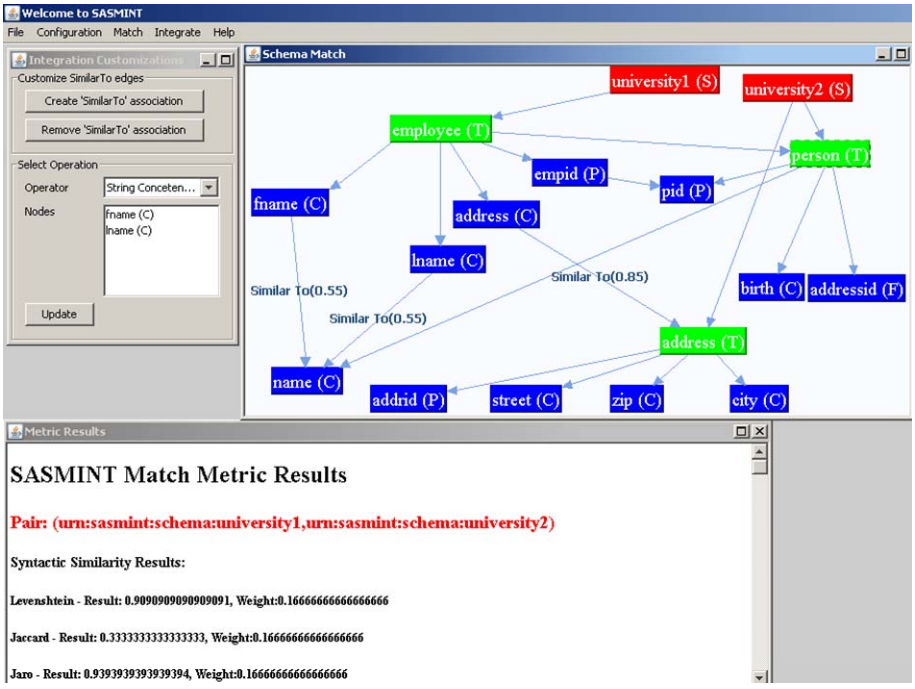

Fig. 5.11. Result of Schema Matching after User Validation

between matching nodes, detailed information about the results of each metric for pairs is also displayed on the GUI. User can modify the results on the graph. He can delete incorrect matches and introduce new ones and specify which kind of operation to use for combining *n* columns in 1-to-n or n-to-1 match cases. When the modifications are completed, the user can save the match results. A portion of the saved XML file for the example schemas is shown in Figure 5.12. This file can be used by a query processor to rewrite the query in terms of local schemas.

```
……..
……..
<graph:snode graph:id="urn:sasmint:column:university1:employee:empid" graph:name="empid“
        graph:schema="university1" graph:table="employee" graph:type="COLUMN“ graph:pkColumn=“yes“ />
<graph:snode graph:id="urn:sasmint:column:university1:employee:fname" graph:name="fname"
        graph:schema="university1" graph:table="employee" graph:type="COLUMN"/>
<graph:snode graph:id="urn:sasmint:column:university1:employee:lname" graph:name="lname"
        graph:schema="university1" graph:table="employee" graph:type="COLUMN"/>
<graph:snode graph:id="urn:sasmint:column:university1:employee:address" graph:name="address"
        graph:schema="university1" graph:table="employee" graph:type="COLUMN"/>
<graph:snode graph:id="urn:sasmint:schema:university2" graph:name="university2" graph:type="SCHEMA"/>
<graph:snode graph:id="urn:sasmint:table:university2:address" graph:name="address" graph:schema="university2"
         graph:type="TABLE"/>
……..
……..
<graph:sedge graph:id="urn:sasmint:hastable:683bb557-0d3d-4d2f-a3b2-840f3065254a“
         graph:sourceNodeId="urn:sasmint:schema:university1“
         graph:targetNodeId="urn:sasmint:table:university1:employee" graph:type="HASTABLE"/>
<graph:sedge graph:id="urn:sasmint:hascolumn:79b02ca1-9572-41af-b9d7-7db0b4d3390f“
         graph:sourceNodeId="urn:sasmint:table:university1:employee“
         graph:targetNodeId="urn:sasmint:column:university1:employee:empid" graph:type="HASCOLUMN"/>
……..
……..
<graph:sedge graph:id="urn:sasmint:similarTo:3a38c020-d184-4f11-8297-d58125f1784d"
         graph:sourceNodeId="urn:sasmint:column:university1:employee:lname"
         graph:targetNodeId="urn:sasmint:column:university2:person:name" graph:type="SIMILARTO">
                <graph:similarity>0.5142156907717128</graph:similarity>
</graph:sedge>
<graph:sedge graph:id="urn:sasmint:similarTo:e6738592-356d-4b97-9841-3e90b2cde8aa"
         graph:sourceNodeId="urn:sasmint:column:university1:employee:fname"
         graph:targetNodeId="urn:sasmint:column:university2:person:name" graph:type="SIMILARTO">
                <<graph:similarity>0.5142156907717128</graph:similarity>

</graph:sedge>
……..
……..
```

Fig. 5.12. Result of Schema Matching in XML format

5.4.4 Integrating Schemas

After validating the results of schema matching, the user can continue with the schema integration step. Schema integration applies a number of heuristic rules, explained in Chapter 4, in order to automatically generate an integrated schema of recipient and donor schemas. For the example university schemas, SASMINT automatically produces the integrated schema shown in Figure 5.13. Both the graph and XML (based on SDML) representations of the integrated schema are generated by SASMINT. The XML representation defines the derivations used to generate the elements of the integrated schema from the elements of input

schemas. User can modify the generated XML result and then save it. Since for integrating schemas of all nodes in a network, two schemas are integrated at a time, the XML file will expand after each integration process, with the definitions of the new nodes and edges as well as their derivations.

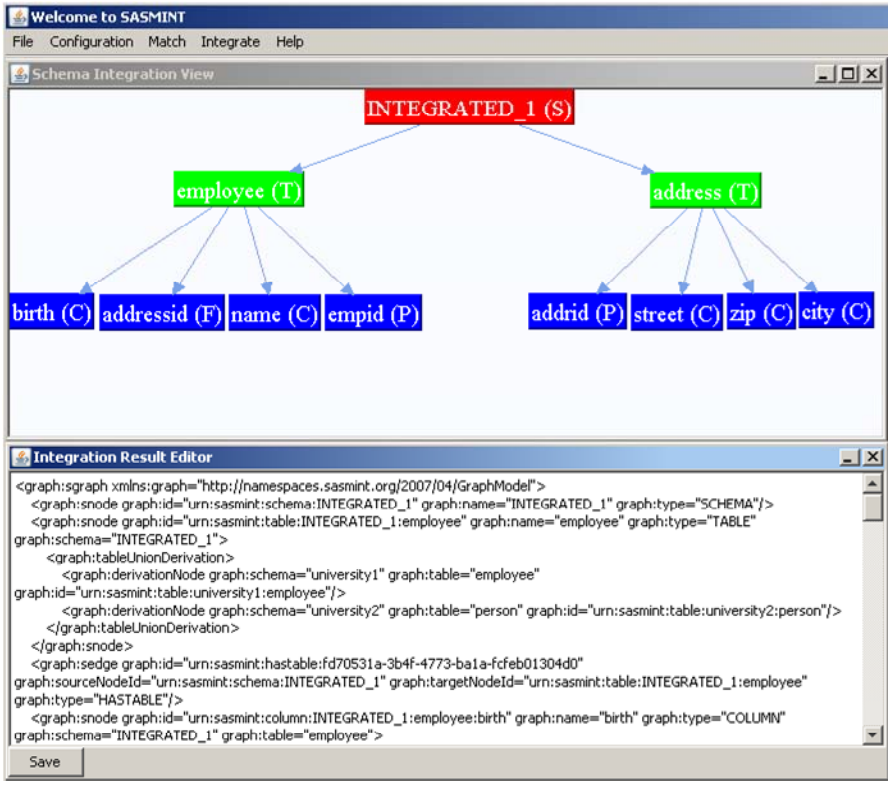

Fig. 5.13. Result of Schema Integration

5.5 Conclusions

In this chapter, we provide the details of the SASMINT implementation, which has been realized in order to both verify and to serve as a proof of concept for our proposed approach. The implementation functionally comprises the processing steps (i.e. called phases), and is made available to the user by means of a GUI, which we consider to be an important aspect of the overall proposed system solution. We observed that most existing schema matching and/or integration tools lacked sophisticated GUI's, which is why we decided to implement a GUI for SASMINT.

The majority of our implementation efforts have been dedicated to implementing 1) The construction of GUI, and 2) The schema integration rules (heuristics)

For implementing the SASMINT system, the platform-independent Java programming language and runtime has been utilized. A number of additional Java libraries, IDE's and add-ons have also been used to enhance our implementation.