Synthesizing Translation Agents For Multi-Disciplinary Data Integration *

Yen-Min Huang

Experimental Systems
IBM Corporation
R.T.P, NC 27715

Chinya V. Ravishankar

Department of EECS
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

Data integration combines data from different data sources, so that an integrated view of retrieved data can be presented to users. Integrating data retrieved from multi-disciplinary data sources can be very complex because many kinds of data heterogeneities must be resolved. Such resolution may be accomplished using translation programs (agents). With a large number of federated databases, developing and maintaining translation software for multi-disciplinary data integration can be prohibitively expensive. Also, delays in translation software development may hinder the availability of newly-created data sets. This paper describes a translation agent synthesis scheme to resolve these difficulties.

Our translation agent synthesis scheme consists of two parts: a formal data set description for characterizing data sets and an agent synthesis mechanism. The data set description describes essential aspects of a data set including schema, types, units, and semantics. The agent synthesis mechanism uses means-ends analysis to construct translation agents. The means-ends analysis is guided by attribute-value lists specifying the needed translation services. An example of inter-disciplinary data integration is provided to illustrate how a translation agent is synthesized.

1 Introduction

This work is motivated by CIESIN's mission. CIESIN (Consortium for International Earth Science Information Network) is participating in a Federally funded project to explore technology necessary to integrate and facilitate the use of global change information. The work includes developing mechanisms to make this information available to scientists, policy-makers, and other user communities in researching and managing global change. CIESIN will serve as a link between relevant databases and an international user community and facilitate access, distribution and use of derived scientific information in the pursuit of understanding and predicting global environmental change [1]. To achieve its mission, CIESIN must develop information management systems which support context-based information retrieval, heterogeneous and distributed databases, and "seamless" user interfaces. One of the challenges is multi-disciplinary data integration.

Multi-disciplinary data integration is difficult in CIESIN's context. First, resolving some types of heterogeneities can be difficult. For example, resolving heterogeneities in data semantics across discipline areas is an open research problem. Second, the data integration must be accomplished without using a global schema, an approach much previous work is based on [2, 3, 4, 5]. This is because the diversity in scientific data makes a single global schema impractical, if not impossible within the CIESIN environment. Finally, developing and maintaining software for data integration can be prohibitively expensive for CIESIN because a large number of data translation programs may be needed to resolve all types of heterogeneities [6].

1.1 Our Approach

Many of these difficulties can be resolved if translation programs (agents) be synthesized either automatically or semi-automatically. A synthesis approach for generating translation agents has many advantages. It can create translation agents on demand, minimizing the costs of software development and maintenance.

*This work was partly supported by the Consortium for International Earth Sciences Information Networking

Also, if translation agents can be synthesized, newly-created data sets can be available for data integration with little delay.

This paper proposes a scheme to synthesize translation agents for facilitating data integration in CIESIN's environment. Our synthesis scheme synthesizes translation agents based on formal data set descriptions, characterizing essential aspects of a data set including schema, types, units and semantics.

The rest of the paper is organized as the follows. Section 2 illustrates CIESIN's data integration through an example. Section 3 describes our data integration model and architecture. Section 4 describes the detail of our agent synthesis scheme. Section 5 concludes the paper and describes future work.

2 A Scenario

We use an example to illustrate some of the data integration issues faced by the CIESIN system developers. Consider the following scenario: a medical scientist is interested in studying how changes in particulate pollution levels has affected the number cases of emphysema in the city of Metropolis over the past few years. The information about emphysema cases is available in a epidemiology database, and the measurements of particulate pollution levels are available in an air-quality database. The relational model is chosen as our platform for this preliminary work. Now, consider a typical query that the scientist may wish to make:

Query: What is the correlation between the measurements of particulate pollution levels and those of emphysema in the city of Metropolis?

To answer this query, two data sets, the epidemiology data set and the air-quality data set, are retrieved from two different data sources (see Tables 1 and 2). The epidemiology data set holds the total number of emphysema cases in the city each year. The air-quality data set provides measurements of the particulate concentrations every six months. However, these two data sets do not provide direct answers to the query mentioned above. The two data sets must be joined to answer the above query, and data translation must be performed as a prelude to the join.

Now, data retrieved from different sources may have different data representations, different data units, and different data semantics. A first step in data integration is data conversion. In our example, the data representation corresponding to the attribute "Year" in the epidemiology data set are very different from that corresponding to the attribute "Date" in the air-quality data set. The data corresponding to the attribute "Year" may be represented as integers (e.g. 1988), while the data corresponding to the attribute "Date" may be represented as strings (e.g. "060188"). The data units of these two attributes are also different (year vs. day-month-year). The data semantics of the two attributes are also different. The attribute "Year" indicates which year the data for the number of emphysema cases belongs to, while the attribute "Date" shows the date on which air quality was measured.

Year	Emphysema Cases
1988	1,052
1989	1,503
1990	2,162

Table 1: Data Set (1) from Epidemiology Database

Date (mmddyy)	Particulate Concentration (gm/liter)
060188	0.00210
120188	0.00220
060189	0.00222
120189	0.00228
060190	0.00232
120190	0.00240

Table 2: Data Set (2) from Air-Quality Database

Data integration must be accomplished by comparing the value corresponding to the "Year" attribute and the value corresponding to the "Date" attribute. However, such data comparison is meaningful only if two data items are on a common referential basis, i.e., their data representations, data units and data semantics are the same. Therefore, data of the "Date" attribute must be converted to the "Year" attribute before two data sets can be integrated. But conversion may also involve *data refinement*. For example, two half-year air-quality measurements may first be averaged to obtain a yearly air-quality value, then the two data sets merged. This is a form of data refinement which usually requires considerable domain knowledge, and it may be necessary to involve human experts in the process. The result for the query may look like Table 3.

Year	Particulate Conc.	Emphysema Cases
1988	0.00215	1,052
1989	0.00225	1,503
1990	0.00236	2,162

Table 3: The Integrated Data Set For the Query

3 Data Integration Model and Architecture

This section will describe our data integration model and architecture which our translation agent synthesis scheme is based on. Data integration is the process of combining data from different data sources, so that an integrated view of retrieved data can be presented to users. The process of combining different data sources may involve repeatedly applying the following steps:

1. **Data Transformation**
   Data transformation is a process which converts data between different data representations, different data types, or different data units. Data transformation does not change the meaning (semantics) of a data set. It simply changes values or representations of data. In other words, data transformation is the process of establishing the common referential basis between two data sets, so that it is meaningful to compare or to merge two data sets.

2. **Data Synthesis**
   Data synthesis refers to the process of creating new data sets with data semantics different from those of existing data sets. For example, data synthesis may involve averaging data points, or joining two data sets. Data synthesis must operate on data that have the same referential basis to produce meaningful results. For example, data should not be averaged unless they are represented using the same unit. Specifically, a data synthesis process is called *data refinement* if it involves only one data set.

The nested relationships between above data integration steps are illustrated in Figure 1. Data integration is achieved by data synthesis processes. However, the data synthesis process may require data transformations to achieve a common referential basis for synthesis. The data transformation may invoke several different types of data conversion to accomplish the transformation process. When the necessary data transformation is completed, the data synthesis process can proceed to synthesize the data. Data integration is complete when all the required data sets are properly synthesized.

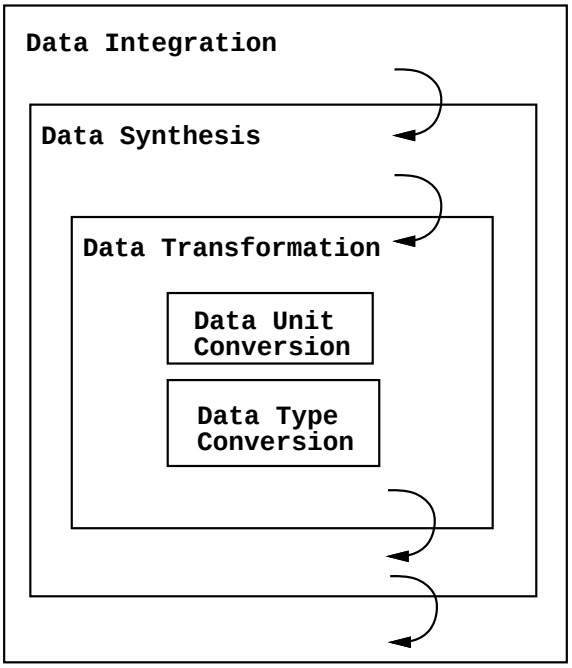

Figure 1: The Relationship Between Data Integration Steps.

3.1 Architecture

The data integration architecture realizes the data integration model. There are two components in the data integration architecture: agents and adapters. Agents are the programs that perform data transformation and synthesis. An agent can be decomposed into smaller units called adapters, which either perform some specific data conversion or provide interfaces to other part of the system. Adapters in an agent are organized as a tower structure (see Figure 2), which reflects the order of data integration steps. Adapters are the basic units used in building an agent, and are either provided directly through library routines or synthesized from other adapters. For complex data integration, multiple agents may be necessary to perform the task. They are organized in a hierarchical structure, and executed in a bottom-up order. Figure 2 illustrates the data integration architecture with multiple agents.

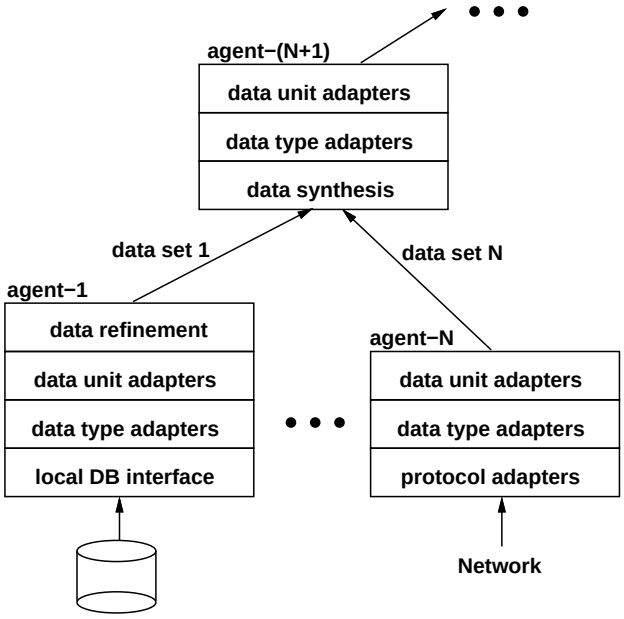

Figure 2: The Data Integration Architecture With Multiple Agents

3.2 Handling the Adapter Proliferation Problem

The adapter proliferation problem arises because $N * (N - 1)$ different adapters are needed to provide directly conversion between N different types of data. This growth is unreasonable, especially when N is large. The adapter proliferation is not a problem in our scheme because adapter libraries are used for conversion between basic data types and between standard data units. Therefore, the number of these basic adapters are limited. The adapters for non-basic data types and non-standard data units are synthesized from these basic adapters. If the number of basic adapters becomes a concern, the proliferation can be easily controlled with some performance penalty for rarely used conversions. This can be accomplished by providing direct conversions to/from a common type. It can be easily shown that with $2 * (N - 1)$ adapters, the conversion can be achieved among N different types with at most two conversion steps. For frequently-used data conversions, extra adapters can be constructed to provide direct conversions, so that the translation performance will be comparable to the direct conversion for the most of the time. Therefore, with $2 * (N - 1) + k$ adapters, we can achieve performance comparable to the optimal performance with $N * (N - 1)$ adapters and a linear growth rate

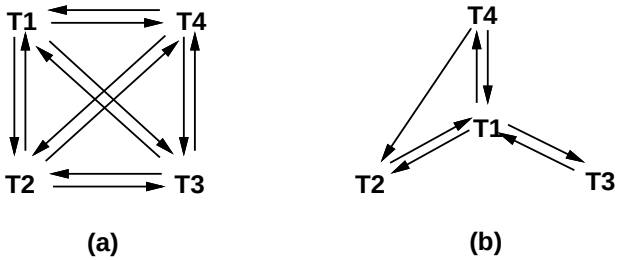

Figure 3: The Maximum Number of Basic Adapters For (a) Direct Conversion (b) At-Most-2-Step Conversion (N=4, k=1)

of adapters at the same time, where k is the number of adapters provided for frequently-used conversions. Figure 3 illustrates both the direct conversion and the at-most-two-step conversion cases for N equal to 4. Each arrow in Figure 3 represents an adapter and the direction of conversion.

4 Translation Agent Synthesis Scheme

Our translation agent synthesis scheme consists of two parts: a formal data set description for characterizing data sets and an agent synthesis mechanism. The data set description covers essential aspects of a data set including schema, units, types, and semantics (Section 4.1). The agent synthesis mechanism combines attribute-naming scheme and means-ends analysis to construct the plan for synthesizing translation agents (Section 4.2).

4.1 Data Set Descriptions

A data set description provides the knowledge necessary for integrating a data set, which must be defined when a data set is exported to the outside world. A data set is assumed to be organized in a tabular or relational format. Data set descriptions are primarily used for facilitating synthesis of translation agents in data integration. The data description specifies four aspects of a data set: the data schema aspect, the data unit aspect, the data type aspect, and the data semantics aspect. Each aspect is represented using the definition language ASN.1 [7]. ASN.1 is chosen here because it is a widely used ISO standard for defining objects and protocols.

4.1.1 The Data Schema Definition

The data schema definition describes all attributes[1] in a data set, and contains the same information as a schema definition in relational databases. It is a sequence of ordered pairs, each consisting of an attribute name and the data type of the attribute. However, the data type can be a complex data type, which is not allowed in traditional relational databases. In ASN.1, the data schema definition is described using SEQUENCE-OF and SEQUENCE[2]. For example, the schema definition for the Air-Quality data set may look like the following, where each entry in the schema has two attributes: "Date" and "ParticulateConcentration", which have the data type "DateString" and "Gram-Per-Liter" respectively.

```
SCHEMA Air-Quality ::= SEQUENCE OF
                                Air-Quality-Entry
Air-Quality-Entry ::= SEQUENCE {
        Date                    DateString,
        ParticulateConcentration Gram-Per-Liter
}
```

4.1.2 The Data Unit Definition

The data unit definition represents the unit that the data is expressed in. Examples of units are seconds, centimeters, kilograms, or number of people. Data units may also be more complex. For example, acceleration may have the units $meter/second^2$. The data unit definition for acceleration may look as follows, where a one-to-one correspondence exists between the data unit definition and its mathematical definition.

```
UNIT Acceleration ::= SEQUENCE {
        meter       LengthUnit,
        divOp       UnitOperator,
        second      TimeUnit,
        expOp       UnitOperator,
        exponent    INTEGER { square(2) }
}
```

It is advantageous to include data unit definitions in data descriptions. Some units are international standards and used across different disciplines uniformly. The units used can provide fundamental semantics about a data set. Also, the number of different unit systems is limited, and the mappings across them are well-understood. Therefore, it is possible to provide a complete set of data conversion routines without excessive development cost.

Data unit definitions are essential to an automatic/semi-automatic data conversion system. They provide unambiguous, mathematical definitions of data, and are quite unlike attribute names in a data set, which are usually too ambiguous and imprecise in meaning to be useful for automatic data conversion. Also, data unit definitions can be used to catalogue data unit conversion routines, so that the process of locating conversion routines can be automated. Each data unit conversion routine performs unit conversion according to the differences between the source and the destination data unit definition. By requiring a unique name for each data unit definition, each unit conversion routine can be uniquely identified by pairing the name of the source and the destination data units. This pairing provides a simple naming mechanism for locating or synthesizing a specific data unit conversion routine (see Section 4.2 for more details).

Another advantage of providing data unit definitions is that they may indicate whether or not two data sets are semantically compatible. Two data sets are semantically compatible if they can be correlated meaningfully. Two data sets are unit-compatible if it is possible to perform data unit conversion between these two data sets. Unit compatibility defines a necessary condition for semantic compatibility. Unit compatibility can be determined by performing text matching on unit names in a data unit thesaurus/dictionary, wherein data units are organized in unit-compatible groups. Once compatibility is confirmed, the required data unit conversion routines can be easily located using the source-destination definition pair.

However, data unit conversion may also be needed between non-standard data units, or those are not defined in our data unit thesaurus/dictionary. When text matching on unit names fails, user intervention is needed. A user can define non-standard units using the existing data units defined in the data unit thesaurus/dictionary. After a non-standard data unit is defined, its definition can be incorporated into the data unit thesaurus/dictionary. This inclusion will allow the system to locate or synthesize the required conversion routines to perform unit conversion to/from this newly-created data unit.

4.1.3 The Data Type Definition

The data type definition defines the data types used in a data schema definition. The data type definition

[1] As in databases, each attribute represents a column within a data set (relation).

[2] An ASN.1 SEQUENCE is analogous to the record structure found in programming languages. SEQUENCE-OF is like SEQUENCE, except that each element in the SEQUENCE must be the same type. For example, SEQUENCE OF INTEGER indicates a series of values, each of which is an INTEGER.

is a physical representation of data which is computer-readable, and is used to catalogue data type conversion routines. Each data type conversion routine is responsible for converting one data type to another. Each conversion routine can be uniquely identified by its source and target data types, and is catalogued on that basis. The data type is recursively defined using ASN.1. For example, the type definition for the data type "DateString" may look as follows, where MonthString ("01" – "12"), DayOfMonthString ("01" – "31"), and Year2DigitString ("00" – "99") are types already defined elsewhere.

```
TYPE DateString ::= SEQUENCE {
        mm    MonthString
        dd    DayOfMonthString
        yy    Year2DigitString
}
```

4.1.4 The Data Semantics Definition

For data integration, the semantics of a data set must be defined. Otherwise, it may not make sense to join two data sets even they have the same data type and data unit definitions. For example, the unit "number of cars" can have many different meanings. It may mean the volume of cars sold, or manufactured, or perhaps the number of cars involved in accidents. It would be a mistake if we try to integrate two data sets solely depending on data type and data unit definitions. Therefore, the meaning (semantics) of two data sets must be known before we integrate them. Data semantics definition helps us define the meaning of data.

The definition of data semantics is also the most difficult among our four data definitions. The definition representation must be general enough to describe all possible data semantics precisely, so that semantic compatibility can be determined automatically. To provide such semantic representation is very difficult, if not impossible. Even if we restrict our domain, no existing methods/representations are likely to be adequate since it appears that we must handle very diverse data sets.

Therefore, we attempt a semi-automatic solution. Our solution is to have the system suggest possible semantics-compatible data sets based on unit compatibility, and let the user make the final decision on whether or not these data sets should be integrated based on the semantics definitions.

The semantics definition is formally viewed as context information for a data set. Each attribute in a data set will be annotated with its semantics definition. In the above example, the context information "car sales" or "cars involved in accidents" will be the annotation for the data attribute "number of cars".

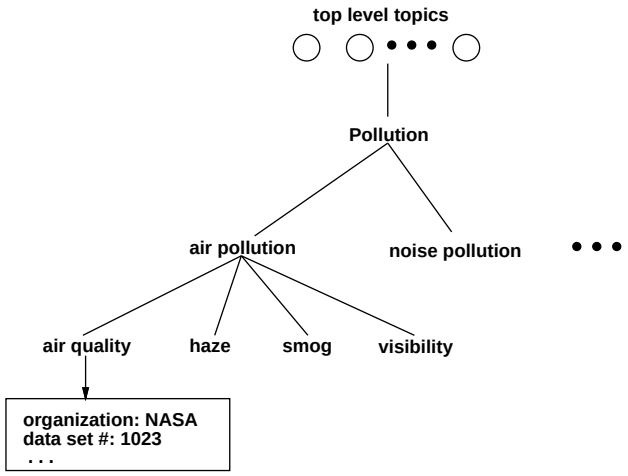

Figure 4: The Global Context Hierarchy.

It is important to have a consensus on how context information is represented and interpreted, so that confusion can be minimized. This is necessary because each data set administrator may have his or her own interpretation of the data set contents. To avoid confusion, administrators must use a common framework to define the semantics of a data set. This common framework represents a level of standardization which must be followed to achieve any shared semantics understanding. Therefore, a global context hierarchy is proposed to achieve this goal.

The global context hierarchy represents a hierarchical classification, where the universe of objects is divided into successively narrower classes. Each node in the hierarchy represents a unique semantic keyword (class), such as *epidemiology* and *emphysema*. The children of each node refine its class. The leaves of the hierarchy contain directory information for the actual data set and its data set description. Figure 4 shows a partial context hierarchy representing a class for air-quality data sets.

To export a data set, a data set administrator navigates the global context hierarchy. The effect of navigation is to place the directory information of a data set at a leaf of the hierarchy and to extract semantics keywords along the navigation path, which become the semantic definition for the data set. The semantics definition is represented as an OBJECT INDENTIFIER in ASN.1. For example, the semantics definition of an air-quality data set in Figure 4 may be { ... ,

pollution, air-pollution, air-quality }.

Clearly, the key to this solution is proposing a sufficiently complete set of keywords and structure in the context hierarchy. A draft list and hierarchy of keywords has been developed by CIESIN [8].

4.2 Adapter and Agent Synthesis

The major task in adapter/agent synthesis is to locate the appropriate adapters and assemble them into an agent with the correct order. This task is accomplished by combining two schemes: attribute-based naming [9, 10] to locate adapters and means-ends analysis [12] to construct the plan for assembling adapters. An attribute-based naming scheme is used because it provides a location-independent abstraction for specifying the services (functionalities) provided by adapters/agents, resulting in a cleaner system design. This approach has been used in the CYGNUS distributed system [11] as its service acquisition mechanism. In our context, each service provided by an agent can be divided into smaller services provided by a set of adapters. Each service is represented by a list of attribute-value pairs. The attribute-value list is used to describe an agent/adapter that needs to be synthesized. The following are two examples using attribute-value lists to represent services.

```
(ServiceClass=Data-Type-Conversion,
      ServiceName=STRING-INTEGER,
      SrcName=Emphysema.year)
(ServiceClass=Data-Synthesis, ServiceName=JOIN,
      SrcName=(Air-Quality.Date, Emphysema.Year))
```

The first list states that the system is looking for a data type conversion service to convert the data of the attribute "Year" in the Emphysema schema from STRING to INTEGER. The second list states that the system wants a data synthesis service to join two data sets based on the "Date" attribute in Air-Quality and the "Year" attribute in the Emphysema data set.

Means-ends analysis is used to derive the correct order to assemble adapters/agents. Means-ends analysis is the problem solving technique used in the General Problem Solver (GPS) [12], and uses the divide-and-conquer strategy to achieve its goal. The goal is recursively decomposed into smaller goals (subgoaling) until they can be achieved by the available operators (means). In our context, when the system encounters a service description (goal), it applies means-ends analysis is to determine whether or not appropriate adapters/agents (operators) are available to provide the service. If appropriate adapters/agents are not available, the system recursively decomposes the current service into smaller services by consulting its synthesis-rule database, which contains the rules for decomposing services. The decomposition process continues until either all the decomposed services are provided by some existing adapters/agents, or the system discovers a service that cannot be satisfied (i.e., a service can neither be decomposed nor be provided by existing adapters/agents). This decomposition process can be represented by a tree structure called the service decomposition tree. Each node in the service decomposition tree represents a service. The service request at a leaf node is satisfied if there exists an adapter/agent can provide this service. The service request at an intermediate node is satisfied if the service requests at all its children are satisfied. After all service requests in a decomposition tree are satisfied, the synthesis plan can be extracted by traversing the decomposition tree and threading all the leaf nodes in order. This order represents the synthesis plan for generating the agen/adapter code. Because each service at a leaf node is satisfied by an existing adapter/agent, the synthesized code will simply consist of a sequence of calls to existing adapters/agents in the order described in the synthesis plan. In the case that some services cannot be satisfied, the user must provide the necessary agents/adapters to satisfy these services. Figure 5 illustrates a service decomposition tree, which is used to synthesize the agent for joining the emphysema and the air-quality data sets described in the scenario section[3].

The system generates the decomposition tree in Figure 5 as follows. First, the system is given a data synthesis request, which is placed at the root of the tree (Node A). The request is to synthesize an agent (JOIN-Agent) to merge two data sets, Emphysema and Air-Quality, by matching the field "Year" in the Emphysema data set and the field "Date" in the Air-Quality data set. Second, to join two data sets having different referential basis, the system divides the service into two smaller services: a data transformation service (Node B) and a data synthesis service (Node C). This decision is based on the rule that a data transformation service must complete before applying a data synthesis operation (i.e., the JOIN operation). Finally, by examining the data type and unit definitions of the source ("Date") and the target ("Year") attributes, the system discovers that the source and the target have different data units and data types. Therefore, the system further divides the transforma-

[3] This join operation alone does not answer the query in Section 2 directly. We still need to compute the yearly average from the result of this operation.

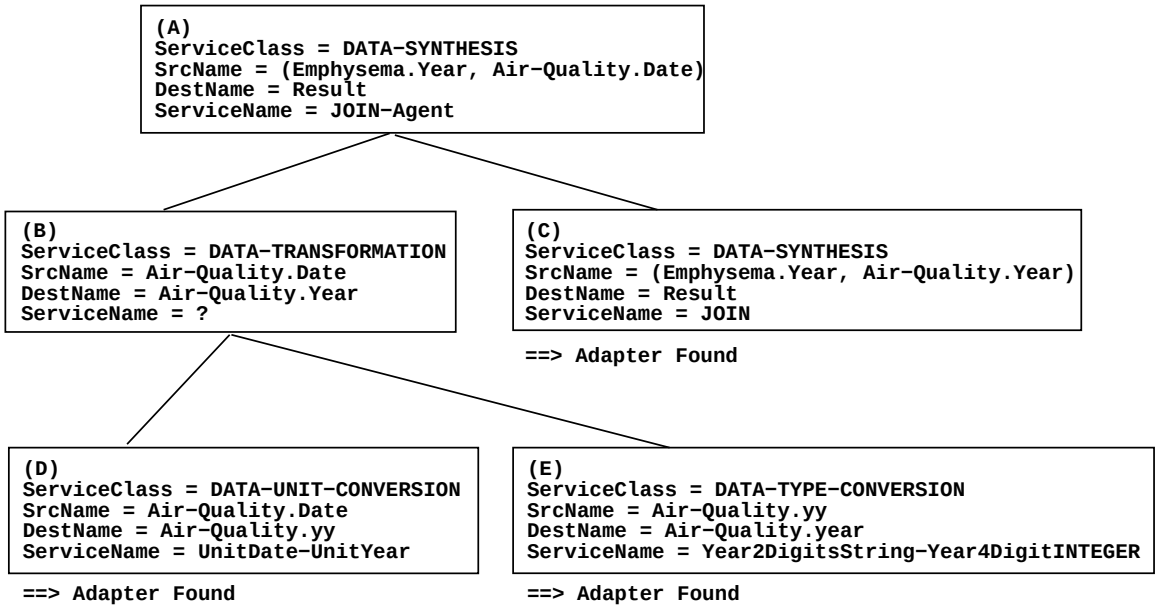

Figure 5: An Example of A Service Decomposition Tree

tion service into a data-unit conversion service (Node D) and a data-type conversion service (Node E). Because both data conversion services can be satisfied by existing adapters, the system completes the process of building this part of the tree, and starts to traverse the other part of the tree. Eventually, all all the services are satisfied, the decomposition tree is ready for the synthesis agents.

After the system completes the process of building the service decomposition tree, the system begins synthesizing the agent code. By traversing the decomposition tree, the following agent code is synthesized:

```
JOIN-Agent(Emphysema,Air-Quality,Result)
{
  NewAttribute( Air-Quality,"yy",
                "Year2DigitString" );
  /* Node D */
  UnitDate-UnitYear(Air-Quality.Date,
                    Air-Quality.yy );
  NewAttribute( Air-Quality,"Year",
                "Year4DigitINTEGER" );
  /* Node E */
  Year2DigitString-Year4DigitINTEGER(
         Air-Quality.yy, Air-Quality.year);
  /* Node C */
  JOIN(Emphysema,Air-Quality,Result);
}
```

The function call "NewAttribute" adds a new attribute in a data set by given an attribute name and the type of the attribute. This function is inserted whenever a new attribute is needed to hold intermediate results. The rest of functions are ordered by traversing the leaf nodes in the service decomposition tree.

5 Conclusions and Future Work

Synthesizing translation agents is advantageous because it can reduce the costs of software development and maintenance significantly. It also can greatly improve the availability of data sets, especially for newly-created ones. It seems feasible in practice and does not require too much software development. This is because the number of adapters are limited and most technologies used in the synthesis scheme have been well-studied and developed.

The contribution of our work is to design a semi-automatic agent synthesis scheme, which can be used for multi-disciplinary data integration. The most design effort is at designing and integrating the following approaches:

- characterizing essential aspects of a data set using formal definition
- using a global context hierarchy to describe data set semantics
- using an attribute naming scheme to describe data integration services and the means-ends analysis to synthesize agents.

Although many issues still need to be studied, our research work is currently focus on automation in semantic schema integration and mechanisms for facilitating data sets discovery [13] and synthesizing protocol adapters [14]. Our development work is currently focus on prototyping our agent synthesis system.

References

[1] N. Roller et al. Draft: Ciesin mission statement. Technical report, CIESIN, June 1991.

[2] J. P. Fry, E. Birss, and et al. An assessment of the technology for data- and program-related conversion. In *Proceedings of AFIPS National Computer Conference*, volume 47, June 1978.

[3] T. Landers and R. Rosenberg. An overview of MULTIBASE. In H. Schneider, editor, *Distributed Data Bases*, pages 153–183. North-Holland, 1982.

[4] A. Sheth and J. Larson. Federated database systems for managing distributed heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3), September 1990.

[5] R. Ahmed and et al. The Pegasus Heterogeneous Multibase System. *IEEE Computer*, 24(12):19–27, December 1991.

[6] D. K. Hsiao and M. N. Kamel. Heterogeneous Databases: Proliferations, Issues and Solutions. *IEEE Trans. on Knowledge and Data Engineering*, 1(1):45–62, 1989.

[7] OSI. Information Processing – Open System Interconnection – Specification of Abstract Syntax Notation One (ASN.1). Technical Report International Standard 8824, International Organization for Standard and International Electrotechnical Committee, 1987.

[8] CIESIN. Ciesin global environmental directory: Phase 1 concept demonstration design document. Technical report, CIESIN, September 1991.

[9] G. Neufeld. Descriptive naming in x.500. In *SIGCOMM '89 Symposium, Communications Architectures and Protocols*, pages 64–71, September 1989.

[10] L. Peterson. The profile naming service. *ACM Transactions on Computer Systems*, 6(4), November 1988.

[11] R. N. Chang and C. V. Ravishankar. A Service Acquisition Mechanism for the Client/Service Model in Cygnus. In *Proc. of 11th International Conference on Distributed Computing Systems*, pages 90–97, May 1991.

[12] A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall Book Company, Englewood Cliffs, NJ., 1973.

[13] N. Hinds, Y. Huang, and C. Ravishankar. A semantic data dictionary method for database schema integration in CIESIN. In *Proceedings of the Conference on Earth and Space Science Information Systems*, Feburary 1992.

[14] Y. Huang and C. V. Ravishankar. Accommodating RPC Heterogeneities Using Automatic Agent Synthesis. Technical report, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan, 1992.