Semantic Information Integration with Software AGs Information Integrator

Jürgen Angele, ontoprise GmbH, Germany
Michael Gesmann, Software AG, Germany

Abstract

For the integration of data that resides in autonomous data sources Software AG uses ontologies. Data source ontologies describe the data sources themselves. Business ontologies provide an integrated view of the data. FLogic rules are used to describe mappings between data objects in data source or business ontologies. Furthermore, FLogic is used as the query language. FLogic rules are perfectly suited to describe the mappings between objects and their properties. Some of these mapping rules can be generated automatically from the data sources metadata. Some patterns do frequently reoccur in user-defined mapping rules, for instance rules which establish inverse object relations or rules which create new object relations based on the objects' property values.

Within our first project access to information is still typical data retrieval and not so much knowledge inference. Therefore, a lot of effort in this project concentrated on query functionality and even more on performance. But these are only first steps. To strengthen this development and to get more experience in this field Software AG recently joined several EU research projects which all have a focus on exploitation of semantic technology with concrete business cases.

Introduction

Data that is essential for a company's successful businesses often resides in a variety of data sources. The reasons for this are manifold, e.g. load distribution or independent development of business processes. But data distribution can lead to inconsistent data which is a problem in the development of new businesses. Thus the consolidation of the spread data as well as giving applications a shared picture of all existing data is an important challenge. The integration of such distributed data is the task of Software AG's "crossvision Information Integrator" one of the components in the crossvision SOA suite [crossvision].

Information Integrator is based on ontologies. On one hand, data source ontologies can be generated from metadata of underlying data sources. Currently, SQL databases, Software AG's Adabas databases, and web services are supported types of data sources. On the other hand, more business oriented ontologies can be developed. These business ontologies make use of other business ontologies or can directly use data source ontologies. FLogic rules describe the information how objects in different ontologies are related to each other.

Using ontologies Information Integrator solves three major problems. First of all it provides all means to integrate different information systems. This means that comfortable tools are available to bring data from different systems together. This is partially already solved by systems like virtual or federated databases [Batini et al. 1986]. Information Integrator is more powerful compared to most of these systems as it not only supports databases but additional sources like web services, applications etc. The second problem which is solved is that Information Integrator allows reinterpretation of the contents of the information sources in business terms and thus makes these contents understandable by ordinary end users and not only by database administrators. Finally this semantic description of the business domain and the powerful mapping means from the data sources to the business ontology solves the semantic integration problem which is seen as the major problem in information integration. It maps the different semantics within the information sources to the shared conceptualization in the business ontology.

Within Software AG Information Integrator was used for a first project Customer Information Gateway (CIG) whose mission was to integrate data that on one side resides in a support information system and on the other side is stored in a customer information system.

Conceptual Layering

Conceptually Information Integrator arranges information and the access to information on four different layers (cf. fig 1):

- The bottom layer represents different data sources which contain or deliver the raw information which is semantically reinterpreted on an upper layer viz. ontologies. Currently relational databases, Adabas databases and web services are supported.
- The second layer assigns a so called "data-source ontology" to each of the data sources. These "data-source ontologies" reflect only database or WSDL schemas of the data sources in terms of ontologies and can be created automatically. Thus they are not real ontologies as they do not represent a shared conceptualization of a domain.
- The third layer represents the business ontology using terminology relevant to business users. This ontology is a real ontology, i.e. it describes the shared conceptualization of the domain at hand. It is a reinterpretation of the data described in the data-source ontologies and thus gives these data a shared semantics. As a consequence a mental effort is necessary for this reengineering of the data source contents which cannot be done automatically.

- On a fourth layer views to the business ontologies are defined. Basically these views query the integration ontology for the needed information. Exposed as Web services they can be consumed by portals, composite applications, business processes or other SOA components.

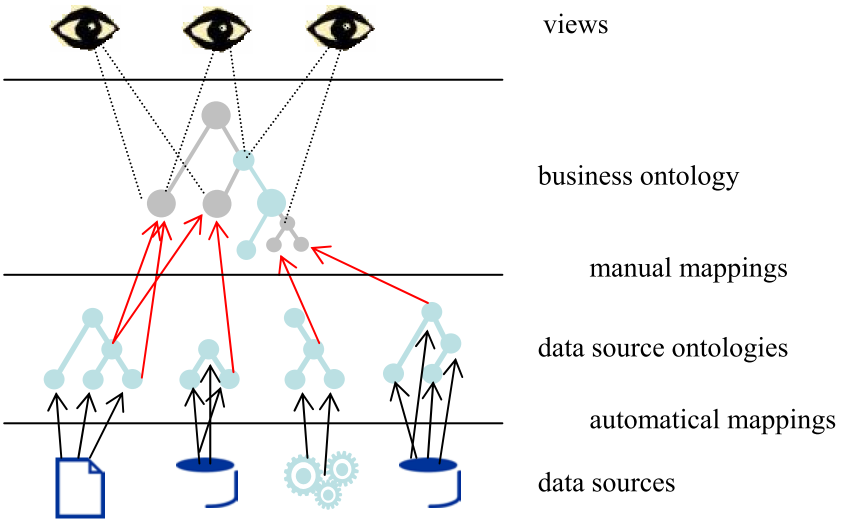

Fig 1. Conceptual Layering of Ontologies

The elements of the different layers are connected by so called mappings. The mappings between the data-sources and the source ontologies are created automatically, the mappings between the ontologies are manually engineered and the views are manually defined queries. Mappings define how source structures are mapped to destination structures. Thus mappings provide ways to restructure information, to rename information or to transform values. Up to now, we do not consider and do not plan to consider approaches which try to automatically derive such mappings [Rahm and Bernstein 2001].

This arrangement of information on different layers and the conceptual representation in ontologies and the mediation between the different models by mappings provide various advantages:

- The reengineered information in the business ontology is a value on its own. It represents a documentation of the contents of the data sources. The representation as an ontology is a medium to be discussed easily by non-IT experts. Thus aggregating data from multiple systems this business ontology provides a single view on relevant information in the user's terminology. More than one business ontology enables different perspectives into the same information.

- It is easy to integrate a new data source with a new data schema into the system. It is sufficient to create a mapping between the corresponding source ontology and the integration ontology and thus does not require any programming know-how; pure modelling is sufficient.

- The mediation of information between data sources and applications via ontologies clearly separate both. Thus changes in the data source schemas do not affect changes in the applications, but only affect changes in the mediation layer, i.e. in the mappings.
- This conceptual structure strongly increases business agility. It makes it very easy to restructure information and thus to react on changing requirements. Neither the data sources nor the applications have to be changed. Only the business ontology and the mappings have to be modified. Again no programming skills are required, all is done on a model level. Thus it minimizes the impact of change, eases maintenance and allows for rapid implementation of new strategies
- Ontologies have powerful means to represent additional knowledge on an abstract level. So for instance by rules the business ontology may be extended by additional knowledge about the domain. Thus the business ontology is a reinterpretation of the data as well as a way to represent complex knowledge interrelating these data. So business rules are directly captured in the information model, they determine the optimal system access and bring every user to the same level of effectiveness and productivity.

Tool Support / Architecture

The crossvision Information Integrator provides a full fledged tool environment for defining models, for mappings between these models and for running queries (cf. fig 2). IntegratorStudio is an ontology engineering environment based on OntoStudio^{TM}.

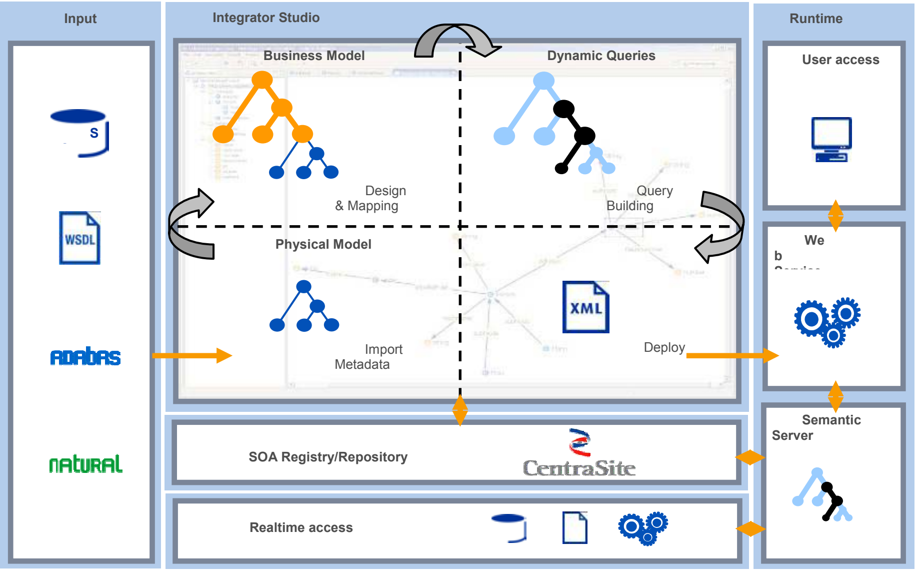

Fig. 2 Architecture of the crossvision Information Integrator

It allows for defining classes with properties, instances of these classes and rules. Import capabilities generate “source ontologies” from underlying data sources. A powerful mapping tool allows user to interactively define mappings between ontologies by graphical and form based means (cf. fig. 3). Rules may be defined with graphical diagrams. IntegratorStudio supports F-Logic, RDF(S), OWL for import and export. Queries which define the mentioned views can be generated and may be exported as web services.

SemanticServer, the reasoning system, provides means for efficient reasoning in F-Logic [Kifer, Lausen, Wu 1995]. SemanticServer performs a mixture of forward and backward chaining based on the dynamic filtering algorithm [Kifer, Lozinskii 1986] to compute (the smallest possible) subset of the model for answering the query. During forward chaining not only single tuples of variable instantiations but sets of such tuples are processed. It is well-known that set-oriented evaluation strategies are much more efficient than tuple oriented ones. The semantics for a set of F-Logic statements is then defined by a transformation process of F-Logic into normal logic (Horn logic with negation) and the well-founded semantics [Van Gelder, Ross, Schlipf 1991] for the resulting set of facts and rules and axioms in normal logic.

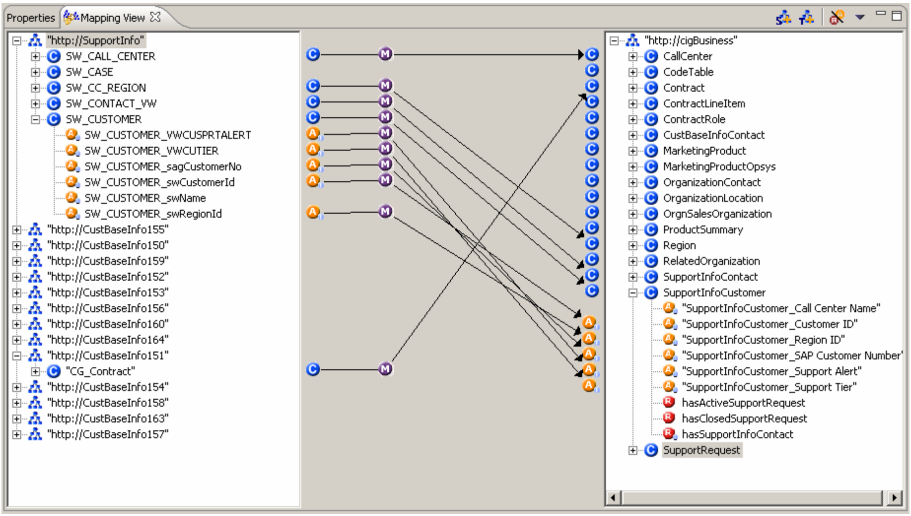

Fig. 3 Mapping Tool in crossvision Information Integrator

Meta data like ontologies, their mappings, web service descriptions and meta information about data sources are stored in the CentraSite repository. Also, IntegratorStudio stores information about exported web services in CentraSite. During startup the inference engine SemanticServer which is based on OntoBroker^{TM} loads the ontologies from the repository and then waits for queries from the exported web services. These queries are evaluated by SemanticServer and are online translated into calls to access connected data sources.

Thus SemanticServer represents the run-time engine, IntegratorStudio the modelling environment and CentraSite the meta data repository. SemanticServer is also integrated into IntegratorStudio thus enables immediate execution of queries to the ontologies.

Use Case: Customer Information Gateway

Within Software AG the Information Integrator was used for a first project whose mission was to integrate data that on one side resides in a support information system and on the other side is stored in a customer information system. The support information system maintains for example information about customers, their contact information and active or closed support requests. The customer information system stores information about clients, contracts etc. While the customer system stores its data in an Adabas database, the support system uses an SQL server. The integrated data view is exposed in a browser based application to various parties inside the company. For instance, instead of only seeing reported support requests support engineers shall get a more complete picture of a customer while talking with them.

For illustration purposes we first sketch a very simplified excerpt of imported data and the business ontology. Throughout the following examples we use FLogic syntax.

First of all there are two classes which have been generated by the mentioned automatic mapping from Adabas files:

```
FILE151_CONTRACT[FILE151_AA=>string;
                 FILE151_AE=>date]@Source151.
FILE152_CLIENT[FILE152_AA=>number;
               FILE152_AB=>string;
               FILE152_AC=>string]@Source152.
```

The cryptic names reflect the internal structure of Adabas files. The names “CONTRACT” and “CLIENT” have been specified by the user during the mapping process. Currently, it is application knowledge that FILE151_AA represents a contract number, FILE151_AE contract end dates, FILE152_AA is a customer id, FILE152_AB a customer name and FILE152_AC the customer address.

In our example we also consider two tables from the SQL database. Table “CUSTOMER” contains columns “id”, “name” and “addr”. Table “CASE” stores support requests with columns “caseId”, “status”, “customerId” and a foreign key “forCustomer”. The resulting classes are

```
CUSTOMER[id=>number;
         name=>string;
         addr=>string]@SourceSQL.
CASE[caseId=>number;
     status=>string;
     customerId=>string;
     forCustomer=>CUSTOMER]@SourceSQL.
```

The business ontology shall contain three classes:

```
Customer[name=>string;
```

```
                address=>string;
                supportRequests=>>SupportRequest]@Business.
SupportRequest[id=>number;
                    status=>string;
                    issuedBy=>Customer]@Business.
Contract[contractId=>string;
               endOfContract=>date;
               endOfContractFormatted=>string]@Business.
```

In the sequel we present some examples on how we used rules within our ontologies and derive some requirements and use cases for rule languages to be used in such a project.

Data source import

The system needs to be open in a sense that it allows for extensions which provide access to external data sources. In Information Integrator so-called built-in predicates implement this. In the sequel we abstract from a concrete syntax of these built-in predicates. Instead we illustrate this by a generic predicate “accessToSource”:

```
 accessToSource(connectionInfo, “tablename”, “rowid1”, X, “rowid2”, Y, …)
```

where *connectionInfo* describes all parameters that are needed to call the data source, *tablename*, *rowid1, rowid2* are names of some database tables or table columns, respectively. *X, Y* are the name of a variables which are to be bound by the built-in predicate.

In our example there are four rules which import data from external data sources:

```
FORALL X,Y
  c(“FILE151”, X):FILE151_CONTRACT[FILE151_AA->X; FILE151_AE->Y]
 <- accessToSource(connectionInfo, “FILE151”, “AA”, X, “AE”, Y)@Source151.

FORALL X, Y, Z c(“FILE152”, X):FILE152_CLIENT
          [FILE152_AA->X; FILE152_AB->Y; FILE152_AC->Z]
   <- accessToSource(connectionInfo, “FILE152”,
                           “AA”, X, “AB”, Y, “AC”, Z) @Source152.

FORALL X, Y, Z
    c(“CUSTOMER”, X):CUSTOMER[id->X; name->Y; addr->Z]
   <- accessToSource(connectionInfo, “CUSTOMER”,
                           “id”, X, “name”, Y, “addr”, Z) ]@SourceSQL.

FORALL X,Y,Z
   c(“CASE”, X):CASE[caseId->X; status->Y; customerId -> Z]
<- accessToSource(connectionInfo, ”CASE”,
                        “caseId”, X, “status”, Y; “customerId”, Z) ]@SourceSQL.
```

Import from data sources is easy, as long as the order of single result objects and the order of property values within a row are not significant. This is valid for imports from SQL and mostly also for import from Adabas. For data sources like web services which expect and return XML documents this is no longer true. For chaining of web services, i.e. the result of one or more web services serves as input for another web service, it is necessary to maintain the structure of the original result documents. Preserving the structure leads to complex rules.

Object and property mapping

It is very easy to define that an object in one model representing the data source is also an object in another model representing the business model. For example a contract object in the customer information system is also a contract object in the business model:

```
FORALL X X:Contract@Business <- X:FILE151_CONTRACT@Source151.
```

Similar rules exist for FILE152_CLIENT and CASE. Like for objects it is also easily possible to specify that a property in one ontology maps to a property in another ontology and that all property values in a first ontology are also values of the mapped property in the second ontology.

```
FORALL X, Y, Z
   X:Contract[contractId->Y; endOfContract->Z]@Business
 <- X:FILE151_CONTRACT[FILE151_AA->Y; FILE151_AE->Z]@Source151.
```

If the underlying data from the external sources contains such information, it is also easily possible to describe that two objects are the same. For example a client in the customer information system and a customer in the support information system represent the same object, if these have the same name and address. Please note, in both systems clients and customers, respectively, have a surrogate values as unique keys. But typically these values are not a viable object identifier across independent data sources. Therefore, we need to identify new identifiers. The example in rule terms:

```
FORALL X, Y, Z
    c(”Customer”, Y,Z):Customer[name->Y;address->Z]@Business
    <- X:CUSTOMER[name->Y; addr->Z]@SourceSQL.

FORALL X, Y, Z
    c(“Customer”, Y, Z):Customer[name->Y; address->Z]@Business
    <- X:FILE152_CLIENT[FILE152_AB->Y; FILE152_AC->Z]@Source152.
```

These simple types of mapping are essential for specification of business ontologies on top of data source or other business ontologies. All of these mapping rules can be described in the Information Integrator with graphical means, i.e. developers do not need to see the FLogic syntax.

Property value mapping

Often similar data that is represented in one way in a first database can be represented in a different way in another database. For example:

- data that is encoded in a single column or field might be scattered across multiple attributes in another database (comma-separated name versus firstname and lastname; encoding of some numeric or boolean bits into a single bit array)
- data with different representation (time and date values as a number, as XML types, as SQL values)

In all these cases it is very helpful to have an extensibility which allows for adding functions that implement necessary transformations.

An example in rule terms:

```
FORALL X, Y Y[endOfContractFormatted->X]@Business
  <- EXISTS Z (Y:Contract[endOfContract->Z]@Business and date2string(Z, X)).
```

where date2string() is a predicate that transforms a date from one presentation into another one.

Object references and more metadata

Every functional model needs to describe relations between objects. Object properties are used to express these relationships in a model. Object identifiers are object property values which reference the object with the identifier.

These properties and property values are similar to foreign keys in relational databases. The foreign key information that is provided with the schema description can be used during generation of the data source model.

An example with a foreign key between table “CUSTOMER” (see above) and table “CASE” in rule terms:

```
FORALL X, Y X[forCustomer->c(“CUSTOMER”, Y)]@SourceSQL
   <- X:CASE[customerId->Y]@SourceSQL.
```

This has to be mapped to the business ontology:

```
FORALL X, Y, Z1, Z2
   X:SupportRequest[issuedBy->c(“Customer”, Z1, Z2)]@Business
   <- X:CASE[forCustomer->Y]@SourceSQL
      and c(“CUSTOMER”,Y)[name->Z1; addr->Z2]@SourceSQL.
```

Also, the inverse reference could be generated. But, while there is a name for the foreign key constraint in SQL databases which can be used for the generation, there is no inverse name in schemas. Therefore the creation of the inverse relation is currently postponed to application development:

```
FORALL X,Y X[supportRequests->>Y]@Business
    <- Y:SupportRequest[issuedBy->X]@Business.
```

Even N:M relationships, which need to be implemented in two 1:N foreign key relations, can be expressed directly.

Queries

The learning of new languages is always a substantial investment, in particular if this involves the learning of new programming paradigms. Having different languages for the modelling and for the querying of ontologies bears the potential for impedance mismatches and causes additional costs. Therefore, rule language and query language should at best be the same. The Information Integrator uses FLogic not only as a language for ontology definitions but also as a query language. Because most application developers in the database area are not very familiar with FLogic and other logic languages the Information Integrator provides a graphical tool to define basic queries.

Like queries in database applications, the queries in our project shall provide some result information. It was not the goal to find all explanations, why the returned results are valid results, nor was it a goal to find all variable bindings that lead to a result.

Using FLogic as a query language leads to some typical database query language requirements, for example:

- User-defined projections on the query result should be possible. Object relationships should be contained in the result. E.g. for one customer having multiple contracts each having contract items, then the query result should contain the information which contract item belongs to which contract within a single result per customer.
- Aggregations over data should be possible (although not yet used within the project).

Performance

Because the integrated view is used an application where e.g. support engineers expect immediate or at least fast answers for even complex requests while talking to a customer, the performance of the rule and query processing is a very important requirement. If responsiveness of the system is not sufficient (e.g. response in less than 2 seconds, which actually is not acceptable in many cases), the whole functionality will not be accepted by its users. This means, systems like the described one can only accept rule languages that allow for efficient processing.

Not surprisingly, experience with the system has shown that efficient processing also to a great extent depends on an optimized rule execution order and caching of intermediate results. Problems that showed up here are very similar to many query optimization problems in database systems.

For example, the data source mapping rules we had shown above always addressed only a single table or file in a database. However, a system that implements access to external data sources only via such single-table access rules will definitely not achieve sufficient performance. The reason for this is that resulting access operations do not make use of the data source's query capabilities like join-operations. Therefore, during query processing it is important to consider which data is stored in which system and to make use of existing indexes, uniqueness of values, or of join capabilities etc.

The current implementation of Information Integrator answers queries all at once. Like in other data intensive applications, it would sometimes be more convenient to have a streaming or cursor result which delivers first results quick and further results on demand.

Summary

A data model in Information Integrator consists of ontologies. Data source models describe the structure of data that resides in external data sources. Business ontologies provide a conceptualization of business entities. FLogic rules are used to specify mappings which assemble higher-value business ontologies from other ontologies. Rules are the first choice to express semantics that is not immediately available within the data. Rules within the ontologies allow to express semantics that otherwise had to be evaluated in queries or applications.

Within our first project, the access to information in these data models is still typical data retrieval and not so much knowledge inference. Therefore, many requirements expressed here are typical requirements for querying in data intensive applications (cursor, performance, query functionality).

With an increasing number of web services where quite some of them simply expose data, we also expect the need to support data integration for such web services. Because web services expect and expose structured data, a rule language should directly support this.

The crossvision Information Integrator based on ontoprise OntoStudio^{TM} and Ontobroker^{TM} is the first step for Software AG in the field of semantic technologies. Recently we joined various EU research projects like NeOn (Lifecycle Support for Networked Ontologies) [NEON], “Business Register Interoperability Throughout Europe” and “SemanticGov: Services for Public Administration” [SemanticGov]. All these projects address concrete business cases. With our participation in these projects we intend to achieve deeper understanding of needs for adequate tooling and runtime systems when using semantics technologies for data integration. On the other hand we will contribute our knowledge about data-intensive processing.

Referenzen

[Batini et al. 1986] Batini C., Lenzerini M., Navathe S.B. *A Comparative Analysis of Methodologies for Database Schema Integration.* ACM Computing Surveys Vol. 18(4):323-364, 1986

[Belkin 1980] N.J. Belkin. *Anomalous states of knowledge as a basis for information retrieval*. The Canadian Journal of Information Science, 5:133--143, 1980.

[crossvision] http://www.softwareag.com/crossvision

[Kifer, Lausen, Wu 1995]. Logical foundations of object-oriented and framebased languages. Journal of the ACM, 42; (1995) 741–843

[Kifer, Lozinskii 1986]. A framework for an efficient implementation of deductive databases. In Proceedings of the 6th Advanced Database Symposium, Tokyo, August (1986) 109–116

[Jaro 1989] M. A. Jaro. *Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida*. Journal of the American Statistical Association 84:414–420, 1989.

[Jaro 1995] M.A. Jaro. *Probabilistic linkage of large public health data files (disc: P687-689)*. Statistics in Medicine 14:491–498, 1995.

[NEON] http://www.neon-project.org

[Rahm and Bernstein 2001] E. Rahm, P. Bernstein. *A survey of approaches to automatic schema matching*, VLDB Journal 10(4):334-350, 2001

[SemanticGov] http://www.semantic-gov.org

[Van Gelder, Ross, Schlipf 1991]. The well-founded semantics for general logic programs. Journal of the ACM, 38(3); July (1991) 620–650