Semantic Interoperability Context, Issues, and Research Directions

Paul Johannesson

Department of Computer and Systems Sciences, Stockholm University, Electrum 230, S-164 40 Kista, Sweden, email: pajo@sisu.se

M. Hasan Jamil
Department of Computer Science, Concordia University, Montreal, Quebec, Canada, email: jamil@cs.concordia.ca

Abstract

An increasing dependence and cooperation between organisations has created a need for many enterprises to access remote as well as local information sources. Thus, it becomes important to be able to interconnect existing, heterogeneous information systems. One form of heterogeneity is semantic heterogeneity, which occurs when there is a disagreement regarding the interpretation and intended use of related information, or when the same phenomenon in a Universe of Discourse is modelled in different ways in two systems. In this paper, we survey the basic problems caused by semantic heterogeneity and suggest a number of research directions that address these problems.

1 Introduction

Interoperability is becoming a critical issue for many organisations today. An increasing dependence and cooperation between organisations has created a need for many enterprises to access remote as well as local information sources. Further, even a single enterprise may have several independent information bases as a result of departmental autonomy. Thus, for many organisations it becomes important to be able to interconnect existing, heterogeneous information systems. One form of heterogeneity is semantic heterogeneity, which occurs when there is a disagreement regarding the interpretation and intended use of related information, or when the same phenomenon in a Universe of Discourse (UoD) is modelled in different ways in two systems. Some examples of semantic heterogeneity are synonyms, homonyms, and scale differences.

Semantic heterogeneity gives rise to many new and unexplored problems in decentralized information systems. As we move from system issues to semantic issues, we move from well defined computational paradigms for symbol manipulation to problems of meaning, and how information is used by different applications and human agents. Inconsistencies, incomplete information, and multiple interpretations of data by different users in different contexts must be handled. A problem here is that in current information systems, much of the semantics of a system resides in the application code rather than in a conceptual schema. To get a complete understanding of the meaning of the data in an information system, it is necessary to study the application programs that manipulate the data. Moreover, the semantics is often not represented directly in the application code, but in the assumptions which the application makes about the data. Such a situation can be accepted in a centralized system in which the applications use a shared set of assumptions, but in a decentralized environment it gives rise to severe problems. It is therefore important to develop methods, tools, and techniques for achieving *semantic interoperability,* i.e., cooperation among semantically heterogeneous systems.

The purpose of this paper is to suggest a number of research directions for the field of semantic interoperability. In the next section, we discuss the contexts in which semantic interoperability is needed. In section 3, we study the causes of semantic heterogeneity and describe various problems that occur in the process of reconciliating semantically heterogeneous systems. In the final section, we propose some directions of research that address the problems in semantic interoperability, including the use of domain knowledge, linguistic knowledge, schema transformations, and machine learning. As many of these research areas are still very immature, we only give a brief outline and do not attempt to present any definite results.

2 Context

2.1 Decentralised Information Systems

Decentralized information systems are becoming increasingly important as a result of organizational demands and as a consequence of technical advances in computer networking. In most approaches to decentralized information systems design, e.g., distributed databases, it has been assumed that the system is viewed by the user as a single logical system described by a global schema, which is centrally maintained. Systems based on these assumptions could, in many cases, become too rigid. A looser form of association could then be preferable. This would consist of independent, autonomous information systems, called nodes, which cooperate in order to exchange information without any global control or a centrally maintained global schema. In such an association, called a *federation* (or a *federated information system*), the nodes themselves decide the conditions for the exchange and sharing of information.

The organizations that manage different information systems are often autonomous, which means that the systems are under separate and independent control. This autonomy gives rise to several types of heterogeneity. Some forms of heterogeneity depend on differences in database management systems, while others are due to differences in the semantics of the information. In order to handle differences between the data models used in different database management systems, several methods for translation between data models have been proposed, [Sheth90]. These methods are used to create mappings between schemas expressed in different data models, for example, the hierarchical, network, relational, and object-oriented data models. However, even when the schemas of different nodes are expressed in the same data model, there may still exist semantic heterogeneities. Work on managing semantic heterogeneity has mainly been carried out within the field of schema integration, [Batini86], [Motro87], [Spaccapietra92]. Schema integration involves merging several schemas into one integrated schema. More precisely, *schema integration* has been defined as „the activity of integrating the schemas of existing or proposed databases into a global, unified schema", [Batini86]. In the context of decentralised information systems, schema integration is often referred to by the term *database integration*.

2.2 Information Systems Development

When analysing a large UoD during the information systems development process, the task can become too complex for a single person or even a small group. To such a task, it is customary to apply a „divide and conquer" strategy. This involves dividing the task of modelling amongst several groups. Each group focuses on a particular part of the UoD and constructs a conceptual schema, called a *local schema*, for that part. When the local schemas have been completed, they are integrated into a *global schema*, which describes the entire UoD. In the context of information systems development, the integration of a number of local schemas into a global one is called *view integration,* which is a special case of schema integration. The problems encountered in view integration are similar to those in database integration. One difference, however, is that in database integration it is required to provide mappings not only on the type level, but also on the instance level between the original and the integrated schemas. This is not required in view integration, unless the original schemas are to be used as views to the integrated schema.

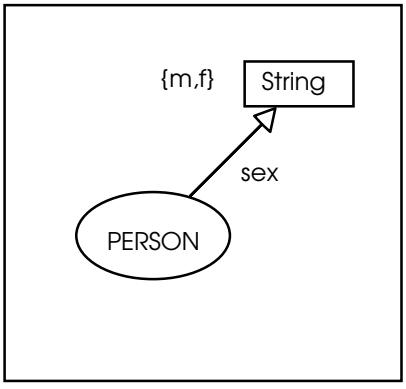
3 Issues

In this section, we discuss the causes of semantic heterogeneity among schemas. We also briefly describe the most important steps in methods for integrating schemas, and in particular we focus on the problem of how to identify and represent similarities between schemas. Finally, we discuss the concept of correctness in schema integration.

3.1 Causes of Schema Diversity

The basic cause of semantic heterogeneity is that a given UoD can be modelled in many different ways. The same phenomenon can be seen from different levels of abstraction, or represented using different properties. More precisely, three types of differences between schemas modelling the same UoD can be identified: differences in terminology, differences in structure, and differences in focus. Differences in terminology arise, as people from different organisations, or from different areas of the same organisation, often refer to the same things using their own terminology. The problematic relationships between terms are of two types: synonyms and homonyms. Synonyms occur when the same object or relationship in the UoD is represented by different names in the two schemas. Homonyms occur when different objects or relationships are represented by the same name in the two schemas. Differences in structure arise when the same aspects of a UoD have been modelled using different constructs. As an example, in a conceptual schema we can classify objects into categories in two different ways. Either a number of subtypes of a given object type can be introduced, or an attribute can be used that has a value indicating the category to which an object belongs. A concrete example is given in fig. 3.1 and fig. 3.2, where the gender of persons in the first schema is represented using an attribute and in the second schema by means of two subtypes.

Differences in focus occur when two schemas describe the same UoD but bring different aspects of it into focus. For example, two schemas may both model people, but one schema may focus on the physical properties of people, such as weight, length, and hair colour, whereas the other schema may describe the social characteristics of a person, such as language, education, and profession.



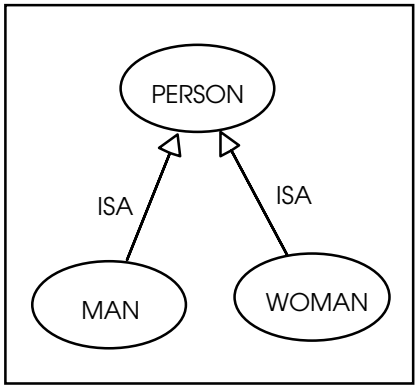


Fig. 3.1

Fig. 3.2

3.2 The Schema Integration Process

In most of the approaches to schema integration in the literature, [Batini86], [Batini91], [Spaccapietra92], the schema integration process is divided into three major phases: schema comparison, schema conforming, and schema merging, see fig. 3.3. *Schema comparison*

involves analysing and comparing schemas in order to determine correspondences, in particular different representations of the same concepts. Three main activities can be identified in this phase: name comparison, structure comparison, and the identification of interschema properties.

Name comparison aims at detecting synonyms and homonyms in the schemas to be integrated. The purpose of *structural comparison* is to identify cases where the same aspects of a UoD have been modelled using different constructs. Name comparison and structural comparison concern relationships between two schemas where constructs are equivalent, i.e. they are able to represent exactly the same information. However, more general correspondences may also exist where constructs appearing in different schemas have certain constraints in common. Such correspondences are commonly called *interschema properties.* An object type in one schema that is a subtype of an object type in another schema is an example of an interschema property, as is an attribute in one schema that is derivable from a set of attributes in another schema.

The aim of *schema conforming* is to transform schemas in order to increase their similarity, which facilitates the schema merging phase. For each synonym or homonym, a renaming is performed. When modelling constructs are semantically equivalent but structurally different, it is impossible to provide any strict rules as to how to make them conform. Usually the schema containing the simpler construct is adjusted by replacing it with the equivalent construct from the other schema.

Schema merging is the final phase in the process of schema integration. It involves merging two schemas into one. Since schema conforming has removed discrepancies in terminology and structure, schema merging becomes a simple superimposition of schemas. Having merged all the local schemas, it may be worthwhile restructuring the integrated schema. It is not unusual once schemas are completely merged, to discover relationships between hitherto unconnected objects. Furthermore, errors that have gone unnoticed in the local schemas might be detected in the integrated schema, in which case the original schemas will need to be reviewed and adjusted. As a consequence, the view integration process becomes iterative, moving back and forth between comparison, merging, restructuring, error analysis, and redesign.

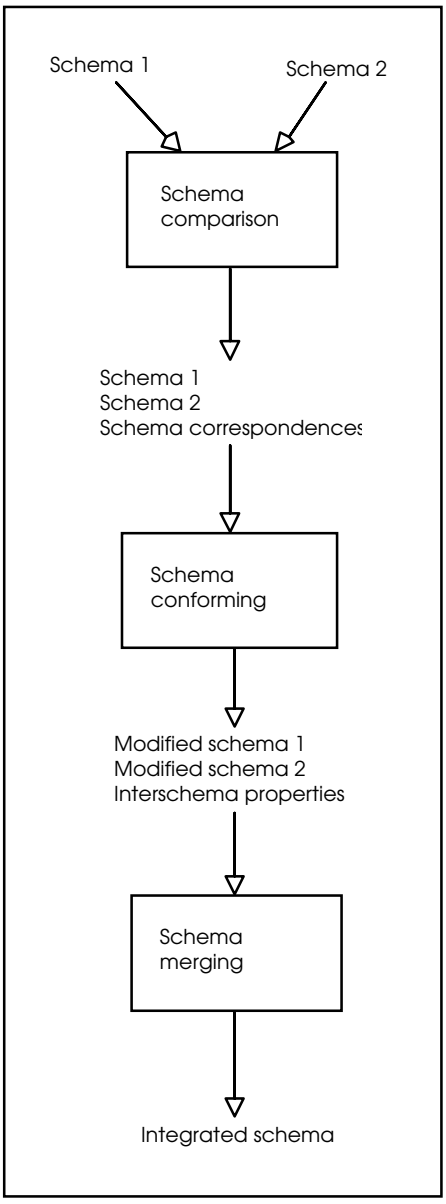


Fig. 3.3 The traditional schema integration process

3.3 Incremental Schema Integration

In most of the approaches to schema integration it has been assumed that two complete schemas are to be integrated at a single occasion. In certain contexts, this may not be a realistic assumption. For example, when a node imports information from another node in a federated information system, the importing node may prefer to integrate only a minor part of the exporting node's schema with its own. As need arises, it may successively integrate larger and larger portions of the other node's schema. In this way, we arrive at an incremental integration process. Whenever a query is submitted, the system explores the nodes in the federation to answer the query and only tries to find out the relevant parts of the nodes. During the query evaluation process, the system builds a component of the global schema. Other parts of the global schema are built subsequently based on other queries. Every time a new component is built, it is integrated with the existing one giving rise to an updated global

schema up to that point in time. A schema constructed in this way is always viewed as incomplete, as there is always a possibility that the next query may add something to the schema.

3.4 Identifying and Representing Schema Correspondences

The most difficult phase in the schema integration process is schema comparison, i.e., identifying semantically related constructs in different schemas. The result of the schema comparison phase is a set of schema correspondences that typically describe set relationships (equality, inclusion, etc.) between the extensions of related constructs in the schemas to be integrated. Schema comparison is an intrinsically difficult problem, since it concerns the semantic interpretation of models created by different human agents. It would, therefore, be futile to attempt to automate the process of schema comparison completely. However, it should be possible to design heuristic methods and tools to assist a user in discovering correspondences between schemas and checking that proposed correspondences are valid. The problem of identifying schema correspondences has as yet not received much attention in the literature. One solution to the problem is suggested in [Bouzeghoub90] and [Comyn92], where it is proposed that correspondences between schemas can be identified by means of a unification process based on similarities between names, structures, and constraints. In the paper [Song92], a method is proposed for discovering schema similarities, utilizing a semantic dictionary and a taxonomy structure. Li and Clifton suggest a pragmatic approach in [Li93], which aims at discovering attribute similarities in a database schema by comparing field specifications. A field specification may include data type constraints, constraints on data values and ranges, keys and foreign keys, access restrictions, etc. In [Fankhauser93], an approach using fuzzy terminological networks is proposed. Semantic resemblance is estimated through the unification of terminology trees that describe the context of an object type. A limitation of most current approaches is that they only consider the information contained in the schemas to be integrated. However, to successfully discover schema correspondences, it is necessary to have access to additional information about the application domain of the schemas; in section 4.1 we shall return to this theme.

There are several different approaches for representing schema correspondences. In the context of federated information systems, the prevalent approach has been to use a common data model (CDM), such as the relational model or an object-oriented model. The individual schemas in a federation are mapped to the CDM, which acts as an interpreter among them. Examples of CDMs are DAPLEX as used in the Multibase project, [Landers82], and the relational model as used in Mermaid, [Templeton87]. A recent approach for representing schema correspondences is to use a higher-order language, such as F-logic, [Kifer90], or SchemaLog, [Lakshmanan93]. The advantage of using a higher-order, logic based language is that it provides clear foundations, a sound formalism, and proof procedures and, thus, offers a declarative environment for schema integration. Conventional

query languages for databases are often based on first order logic and make it possible to query the data in a database. In schema integration, however, it is necessary to be able to query and manipulate also the schema of a database. By using a higher-order language that treats components of meta-data as „first class citizens" it becomes possible to treat data as well as their schema in a uniform way.

3.5 Information Capacity

An important issue in schema integration is to ensure that the integrated schema is, in some sense, correct. Speaking loosely, this means that the integrated schema should be capable of representing the same information as the local schemas being integrated. Most of the approaches to schema integration have not paid much attention to the question of correctness; at best, they provide some informal notion of correctness and state without proof that the integrated schema preserves the meaning of the original ones. Recently, however, formal work on schema integration and transformations have appeared that use the concept of information capacity to judge the correctness of transformed and integrated schemas [Johannesson93d], [Miller93], [Rosenthal87], [Markowitz92].

Two schemas can be compared based on *information capacity,* where, informally, a schema S2 has more information capacity than another schema S1 if every instance of S1 can be mapped to an instance of S2 without loss of information, [Miller93]. In particular, every instance of S1 must be recoverable from its image under the mapping. More precisely, we say that a schema S2 *dominates* (has more information capacity than) a schema S1 if there exists a total, injective function f: I(S1) § I(S2), where I(S1) denotes the instances of S1. S1 and S2 are *equivalent* if f is a bijection. Usually it is required that f be expressible by a query in some query language.

The notion of information capacity can be used to formulate constraints on a schema that is to be used as a view to another schema. Suppose we want to use a schema S1 as an interface to a database stored under another schema S2. If we are to be able to query and view the entire database of S2 through S1 then S1 must dominate S2, as noted by [Miller93]. This simple result gives rise to an interesting observation: the integrated schema obtained in view integration should satisfy other constraints than the integrated schema obtained in database integration. In the case of database integration, the integrated schema works as an interface to the union of the original schemas, and it should thus dominate them. In view integration, on the other hand, the original schemas work as views on the integrated schema, and therefore their union should dominate it. The notion of information capacity is also useful to determine if two constructs from different schemas can be meaningfully integrated, and it, thus, plays a role in the schema comparison phase of the integration process; we return to this topic in section 4.4.

4 Research Directions

4.1 Domain Knowledge

As discussed in section 3.4, the most challenging part of the schema integration process is schema comparison. One way to facilitate schema comparison is to use *domain abstractions* as a vehicle for identifying correspondences between schemas. A domain abstraction is a small, abstract schema that describes the generic features of a type of situation that may occur in many different contexts. The instantiations of a domain abstraction can, therefore, be parts of the schemas of information systems of different types. An example of a domain abstraction is a provider transferring an acquisition to a recipient, and an instantiation of this could be a library lending a book to a borrower. In the area of software engineering, in particular software reuse, domain abstractions have been utilized in several projects, e.g. clichés [Reubenstein91], reusable patterns [Biggerstaff87], and generalized application frames [Constantopoulos92]. The purpose of using domain abstractions in software reuse is to obtain a classification of frequently occurring structures, which facilitates the search for software components that may be reused. Domain abstractions can take many different forms, such as the canonical conceptual graphs of Sowa [Sowa84], hierarchical models of natural categories [Rosch91], scripts [Schank82], and structure mapping theory [Gentner83].

When using domain knowledge in schema integration, we envisage that the conceptual schemas to be compared will first be related to a common set of domain abstractions. Constructs related to the same domain abstraction will then become candidates for comparison and eventually merged in later steps of the schema integration process. This use of domain knowledge will be particularly valuable in the context of federated information systems. In a federation with n nodes, we would have to perform n^2 comparisons if all comparisons are to be made bilaterally; given a set of common domain abstractions, the number of comparisons will be reduced to n.

4.2 Linguistic Approaches

Differences in terminology is one of the main problems when comparing schemas. One way to address this problem is to standardise the terminology used in the schemas to be integrated. A linguistic theory that seems to be applicable to this task is the theory of *case grammars*, [Sowa84]. A case grammar provides a categorisation of noun phrases according to the conceptual role they play in the action described by a sentence. Examples of cases included in most case grammars are the following: agent: one who performs an action; object: the one who suffers an action; recipient: the one for whom an action is performed; instrument: an object used to perform an action; location: the place where an action is performed. In conceptual schemas, the object types correspond to either nouns or verbs, while the attributes correspond to cases. Case grammars provide a set of established labels for attributes in a schema, which help designers to choose meaningful and standardised names when modelling object structures. Describing all associations between objects by a small number of case relations standardizes the terminology of the schemas

to be integrated, and thus facilitates schema comparison by reducing the number of synonyms occurring in the schemas. In [Johannesson93b], we outline how to compare schemas expressed in the ECG formalism [Moulin92], which is based on case grammar.

4.3 Schema Transformations

An important technique in schema integration is the use of schema transformations. A schema transformation maps schemas to schemas; more precisely, a *schema transformation* is a total function F: S1 § S2, where S1 and S2 are sets of schemas. In the schema integration process, schema transformations are used in the schema conforming phase, where semantically equivalent but syntactically different constructs in the schemas are transformed so that they become easy to merge. Typical schema transformations are converting an attribute into an object type or introducing additional subtypes.

An alternative way to use schema transformations, as suggested in [Johannesson93c] and [Johannesson94], is to transform the schemas before schema comparison and thereby facilitate this phase. The schemas should be transformed so that any relationship between them is expressible as a simple relationship between two object types or two attributes. So, the goal is to try to standardize the schemas by transforming them into some „normal form" before integrating them. In this way, we reverse comparison and modification by requiring that the schemas first be modified in order to facilitate the schema comparison. As an example of structural differences, consider fig 3.1 and 3.2, where the gender of people are modelled by means of an attribute and by subtypes, respectively. To avoid discrepancies of this type between schemas, one can require that attributes which can take on only a small number of fixed values should not be allowed. If a schema should contain such an attribute, it could be transformed by introducing a number of subtypes to the domain of the attribute.

In addition to removing structural differences, it is also important to standardize the terminology of schemas as outlined in the previous section. Combining structural and terminological standardisation, we arrive at an alternative schema integration process consisting of the steps depicted in fig. 6.1.

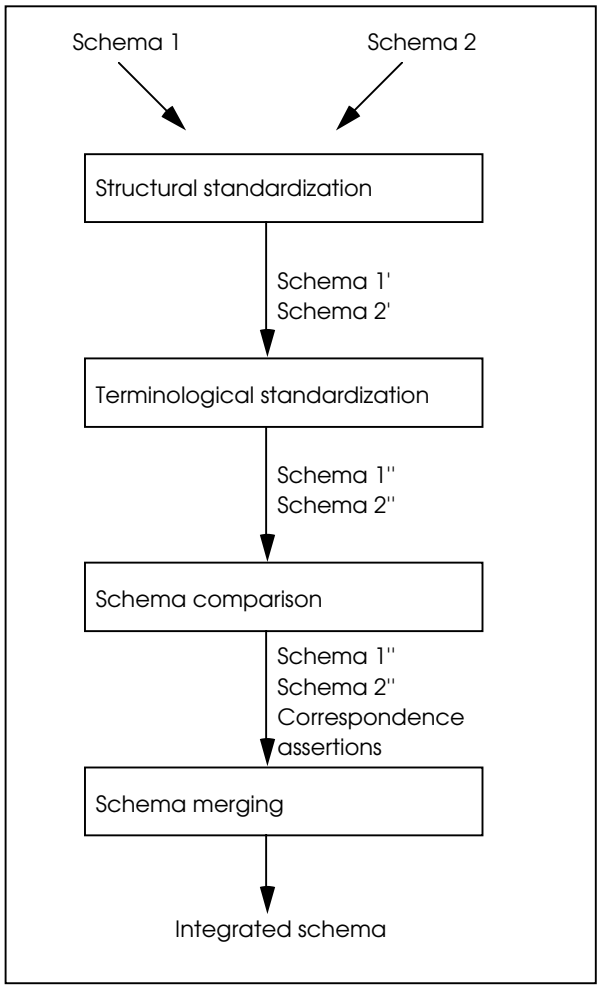


Fig. 6.1 An alternative schema integration process

4.4 Schema Conflicts

Given two conceptual schemas and a set of schema correspondences, it is possible that these are conflicting, i.e., the schema correspondences are not compatible with the semantics of the schemas. Informally, we say they are conflicting if there exists an information base for one of the schemas, and there does not exist an information base for the other schema, such that the schema correspondences hold. As an example, there is a conflict if two attributes, *a* and *a'*, of two different schemas are asserted to be equal, and there is a constraint saying that *a* is single-valued and another constraint saying that *a'* is multi-valued. When suggesting a set of schema correspondences for two schemas, there is always a risk that some spurious correspondence are included. It is, therefore, important to check for conflicts in order to ensure that a proposed set of schema correspondences is reasonable. However, it can be shown that in the presence of general integrity constraints, the problem of determining whether two schemas and a set of schema correspondences are conflicting is undecidable, [Biskup86], [Miller93]. This entails that it is meaningful to address the problem of schema conflicts only for a restricted range of integrity constraints. In [Johannesson92] a set of basic types of integrity constraints is identified, and conditions are given for schema conflicts when only these kinds of

constraints are considered. The types of constraints taken into account are mainly cardinality constraints for attributes and certain simple dynamic constraints. Further work is needed to extend the range of constraints taken into account without sacrificing decidability.

4.5 Machine Learning

Knowledge discovery and learning from databases are emerging areas of research. Many database researchers have proposed ideas and applications of machine learning in artificial intelligence to improve database performance, query optimization, discover hidden relationships of data, learn new properties of objects,etc. So far no attempts have been made to use these techniques in the area of interoperability across databases. However, in the formal front, non-monotonic reasoning, induction, abduction, etc. shows promise for machine learning, but a satisfactory breakthrough is far from reality. Due to lack of a sound formal foundation of knowledge discovery and learning, the approaches that are followed in practice in general are ad hoc. That is they are some kind of application specific solutions or methods.

Most often, learning or discovery is viewed as a task. Usually these tasks are some form of rules that are to be learned or verified -- such as functional and inclusion dependencies, constraints, classification, characteristic rules, etc. Of particular interest, a characteristic rule is an assertion which characterizes the concept satisfied by all of the data stored in the database. For example, the symptoms of a specific disease can be summarized as a characteristic rule. On the other hand, a classification rule is an assertion which discriminates the concepts of one class from other classes. For example, to distinguish one disease from others, a classification rule should summarize the symptoms that discriminate this disease from others, [Cai90]. Learning has been characterized in [Cai90] as a triple <D, C, ¬>, where D represents the set of data that are relevant to a specific learning task, C represents a set of conceptual bias (e.g., generalization hierarchy, meta-data, etc.) useful for defining particular concepts, and ¬ is a language used to phrase definitions.

Usually the set of conceptual biases are supplied by some domain expert. However, these biases may also be learned or discovered from the databases. Unfortunately, in a federated database system, several kinds of heterogeneities makes such learning tasks harder. Uncertainty of learned conceptual biases as a consequence of semantic heterogeneity and incompleteness of expert knowledge becomes a major bottle-neck that needs to be addressed satisfactorily. One way to deal with this issue is to associate a reliability to each piece of knowledge from experts and learned rules, and to adjust their reliabilities according to new evidences, and by using the most reliable knowledge and rules to make the best possible conclusions. For example, it is possible to learn the set of schema correspondences that are proposed in [Johannesson93a] using machine learning techniques if the meta knowledge required is given and

certain. The meta knowledge is the information about the domain, names of relations and attributes, conflicts, similarities, etc. that is required to be given and certain. It should be easy to see that if the knowledge is uncertain, then the learned schema correspondences are also uncertain. Moreover, partial absence of such meta knowledge will result in an incomplete set of integration assertions that leads to a potential danger of inconsistent or incomplete schema integration.

As another example, consider a chain store which have outlets at different provinces in a country which has different tax laws. Now consider a scheme for the inventory relation for every outlet as follows -- *INVENTORY = <{Item, Price, Quantity>}, {Item § Price, Quantity}>.* If some of the outlets decides to include the sales tax in the price of an item, then in the absence of this knowledge, based on the database instances a learner may discover that *Item, Price* or *Item, Price, Quantity* is the key in the integrated schema. Since it is unlikely to enumerate a priori all possible learning tasks, such as this one, as required by the characterization of learning in [Cai90], unwanted or invalid discoveries are possible. A possible way to deal with this kind of situation is to use a method based on *learning from mistakes through explaining*. For example, consider a query on the global schema -- what is the price of the item „crest toothpaste"? The system's response may be a set of tuples giving several prices from several different databases across the country. At this point, the system may be informed that it is not an acceptable answer because *Item § Price* must hold. This may trigger a learning task for the system to figure out whether there exists a *Price* and a set of taxes Tax_i in every database *i* such that *Price ł* Tax_i ,i 1 ¯ 1 ¯ n satisfies the constraint *Item § Price* , where *n* is the number of databases and Tax_is are factors. (Notice that for any item *x,* if the provincial tax is Tax_i, then for each store *i,* Tax_i *ł p = Price*$_i$*.* Hence *p* is the real price of the item *x*.) If the explanation can be validated, then the system may integrate this knowledge in the database and use in future for other purposes. This meta knowledge may also be used to make conversions as well by a context mediator [Sciore92]. Notice that, the system not only improved its learned rule in this way, it also learned a new knowledge about the tax laws in the provinces that may help other conclusions. Other examples of this type of learning are unit mismatch, scale difference, etc.

A related issue that deserves attention is knowledge maintenance that are learned from databases. For any database and its associated scheme, constraints and meta data, called the structure, the extension of the database is an instance of its structure. Although the structure may remain static, the instance may change. Usually the rules that are learned are based on instances, and are „specialized constraints" of its structure that hold on the specific instance of the database. Any change in the instance may invalidate some of the induced rules. It is more so in the case of federated databases, and in the presence of uncertainty in the database knowledge and extensions. Hence a scheme for rule maintenance is essential. Since

the cost of learning is much larger than the cost of updating a rule following a database modification, an *update oriented rule maintenance* seems appropriate. A solution based on a trigger like mechanism may be interesting and cost effective.

4.6 Extending the Object Perspective

Research on schema integration has hitherto almost exclusively focussed on the data, or object, perspective of a UoD. Only the static, structural aspects have been taken into account, together with certain simple types of constraints, such as cardinality constraints. However, when constructing a requirements specification for an information system, many other aspects of the UoD under consideration have to be explicitly represented. In addition to the static aspects of a UoD, it is also necessary to model its dynamics, i.e., how the system evolves over time. Further, the goals of the organisation using the information system must be represented. The activities and functions of the organisation should also be described, as well as the agents carrying out the activities and the structure of responsibility.

A possible extension of the work on schema integration is, thus, to take into account dynamics, goal, activity, and agent perspectives in addition to the static object structures. Such an extension can take two different forms. First, one could construct methods and tools for integrating models representing goal, activity, and agent structures. More concretely, this could mean integrating entity life cycle diagrams, Jackson structure diagrams, dataflow diagrams, etc. An alternative, less ambitious, approach is to still only integrate object structures, but use the other perspectives to facilitate the schema integration process and to ensure that no erroneous integration is made. As an example, if two object types are equal, one should check that the same goals apply to them, and that they are created, changed, and destroyed by the same events. This can be viewed as a way of checking for conflicts, where not only static constraints typically found in database schemas are considered, but also constraints representing other aspects of a UoD.

4.7 Tool Support

A tool for supporting all the phases of the schema integration process will consist of several interrelated components, see fig. 4.2. The central component is the Schema Correspondence Manager that acts as a book-keeper of the schema correspondences that are detected during the integration process. The schema correspondences can either be directly identified and entered by a user, or they can be suggested to him by the Schema Correspondence Identifier module. This module may itself consist of several components that utilise different types of knowledge to find potential schema correspondences. One component may look for name similarities using string matching techniques,, another component may search for synonyms using dictionaries and thesauruses, and a third component may analyse actual database instances. Whenever a new schema correspondence is suggested, the Conflict Analyser checks that it does not give rise to a conflict. As noted in section 4.4, it is not

possible to construct a Conflict Analyser that detects all possible conflicts, but an automatic control of simple and common conflicts is nevertheless valuable. The Schema Transformation Manager, coupled to the Schema Editor, assists a user in rearranging the schemas so that they become easier to merge. Based on the schema correspondences identified and the transformed schemas, the Schema Merger will automatically produce an integrated schema.

An important requirement for a schema integration tool is *flexibility* which, in this context, means that a user should not be forced to adhere to any particular integration method, [Rosenthal91]. A user should be able to control the order of the integration tasks and be allowed to leave some tasks uncompleted while proceeding to other tasks. For example, it should be possible to merge parts of the schemas even if not all schema correspondences have been identified. Another example of flexibility is that the Schema Transformation Manager should support schema standardisation as well as schema conforming.

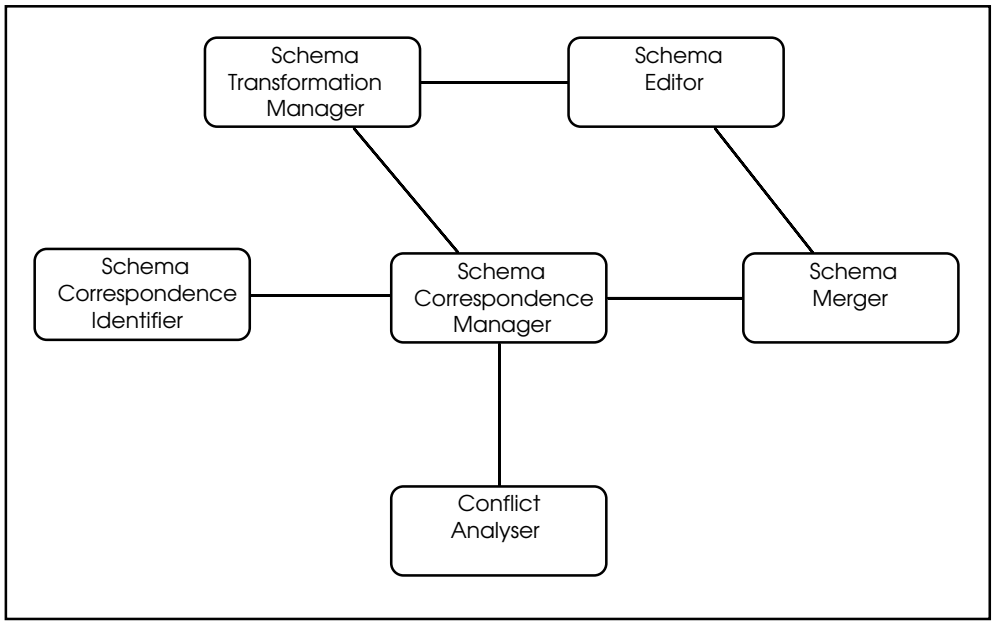


Fig. 4.2 Architecture for Schema Integration Support Tool

References

[Batini86] C. Batini, M. Lenzerini and S. B. Navathe, „A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, vol. 18, no. 4, pp. 323–364, 1986.

[Batini91] C. Batini, S. Ceri, and S. Navathe, *Conceptual Database Design*, Benjamin/Cummings, 1991

[Biggerstaff87] T. Biggerstaff and C. Richter, „Reusability Framework, Assessment, and Directions", *IEEE Software*, March, 1987.

[Biskup86] J. Biskup and B. Convent, „A Formal View Integration Method", in *International Conference on the Management of Data,* Washington, ACM, 1986.

[Bouzeghoub90] M. Bouzeghoub and I. Comyn-Wattiau, „View Integration by Semantic Unification and Transformation of Data Structures", in *Ninth International Conference on Entity-Relationship Approach,* Ed. H. Kangassalo, pp. 413-430, Lausanne, North-Holland, 1990.

[Cai90] Y. Cai, N. Cercone, and J. Han, „An Attribute Oriented Approach for Learning Classification Rules from Relational Databases", *IEEE Data Engineering,* pp. 281-288, 1990.

[Comyn92] I. Comyn-Wattiau and M. Bouzhegoub, „Constraint Confrontation: An Important Step in View Integration", in *4th European Conference on Computer Aided Information Systems Engineering,* Manchester, Springer, 1992.

[Constantopoulos92] P. Constantopoulos, M. Jarke, J. Mylopoulos and Y. Vassiliou, „Software Information Base: A Server for Reuse", ICS-Forth, Greece, 1992.

[Fankhauser93] P. Fankhauser and E. Neuhold, „Knowledge Based Integration of Heterogeneous Databases", in *Interoperable Database Systems,* IFIP Transactions A-25, Ed. D. Hsiao et.al., North-Holland, 1993.

[Gentner83] D. Gentner, „Structure Mapping: A Theoretical Framework for Analogy", *Cognitive Science*, vol. 5, no. pp. 121-152, 1983.

[Johannesson92] P. Johannesson, „Supporting Schema Integration by Rule Analysis", Department of Computer and Systems Sciences, Stockholm University, 1992.

[Johannesson93a] P. Johannesson, „A Logical Basis for Schema Integration", in *Third International Workshop on Research Issues in Data Engineering - Interoperability in Multidatabase Systems,* Ed. H. Schek, Vienna, IEEE Press, 1993.

[Johannesson93b] P. Johannesson, „Using Conceptual Graph Theory to Support Schema Integration", *12th International Conference on Entity-Relationship Approach,* Ed. R. Elmasri, Houston, OmniPress, 1993

[Johannesson93c] P. Johannesson, „Schema Transformations as an Aid in View Integration", in *5th International Conference on Computer Aided Information Systems Engineering,* Ed. C. Rolland, Paris, Springer, 1993.

[Johannesson93d] P. Johannesson, *Schema Integration, Schema Translation, and Interoperability in Federated Information Systems,* PhD-thesis, Stockholm University, 1993.

[Johannesson93c] P. Johannesson, „Schema Standardization as an Aid in View Integration", to appear in *Information Systems,* 1994.

[Kifer90] M. Kifer, G. Lausen, and J. Wu, „Logical Foundations of Frame-based Languages", Technical Report 90/14, SUNY at Stony Brook, 1990.

[Lakshmanan93] L. Lakshmanan, F. Sadri, and I. Subramanian, „On the Logical Foundations of Schema Integration and Evolution in Heterogeneous Database Systems, in *Third International Conference on Deductive and Object-Oriented Databases,* Arizona, 1993.

[Landers82[T. Landers and R. Rosenberg, „An Overview of Multibase", *Distributed Databases,* pp. 153-184, 1982

[Li93] W. Li and C. Clifton, „Using Field Specifications to Determine Attribute Equivalence in Heterogeneous Databases", in *Third International Workshop on Research Issues in Data Engineering - Interoperability in Multidatabase Systems,* Ed. H. Schek, Vienna, IEEE Press, 1993.

[Litwin86] W. Litwin and A. Abdellatif, „Multidatabase Interoperability", *IEEE Computer*,, December, 1986.

[Markowitz92] V. Markowitz and A. Shoshani, „Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach", ACM TRansactions on Database Systems, vol 17, no 3, pp. 423-464, 1992.

[Miller93] R. Miller, Y. Ioannidis, and R. Ramakrishnan, „The Use of Information Capacity in Schema Integration and Translation", *19th VLDB Conference,* Dublin, 1993.

[Motro87] A. Motro, „Superviews: Virtual Integration of Multiple Databases", *IEEE Transactions on Software Engineering*, vol. 13, no. 7, pp. 785-798, 1987.

[Moulin92] B. Moulin and P. Creasy, „Extending the Conceptual Graph Approach for Data Conceptual Modelling", *Data and Knowledge Engineering*, vol. 8, no. 3, pp. 223-248, 1992.

[Reubenstein91] H. Reubenstein and R. Waters, „The Requirements Apprentice: Automated Assistance for Requirements Acquistion", *IEEE Transactions on Software Engineering*, pp. 226-240, March 1991.

[Rosch91] E. Rosch, „Prototype Classification and Logical Classification: The Two Systems", in *New Trends in Conceptual Representation*, Ed. E. Scholnick, LEA, 1991.

[Rosenthal87] A. Rosenthal and D. Reiner, „Theoretical Sound Transformations for Practical Database Design", inn Eighth International Conference on ER approach, Ed. S. March, pp. 115–131, 1987.

[Rosenthal91] A. Rosenthal and M. Siegel, „Building Flexible, Extensible Tools for Metadatabase Integration", *Workshop on Information Technology and Systems,* Cambridge MA, 1991.

[Schank82] R. Schank, *Dynamic Memory: A Theory of Reminding and Learning in People and Computers*, Cambridge University Press, 1982.

[Sciore92] E. Sciore, M. Siegel, and A. Rosenthal, „Using Semantic Values to Facilitate Interoperability among heterogeneous Information Systems", submitted to *ACM TODS,* 1992.

[Sheth90] A. P. Sheth and J. A. Larson, „Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–236 1990.

[Song92] W. W. Song, P. Johannesson and J. A. Bubenko Jr, „Semantic Similarity Relations in Schema Integration", in *11th International Conference on the Entity-Relationship Approach,* Ed. A. M. Tjoa, Karlsruhe, Germany, 1992.

[Sowa84] J. F. Sowa, *Conceptual Structures – Information Processing in Mind and Machine*, Addison-Wesley, 1984.

[Spaccapietra92] S. Spaccapietra, C. Parent and Y. Dupont, „Model Independent Assertions for Integration of Heterogeneous Schemas", *The VLDB Journal*, vol. 1, no. 2, pp. 81–126, 1992.

[Templeton87] M. Templeton et. al., „Mermaid: A Front-end to Distributed Heterogeneous Databases, *Proceedings of IEEE 75,* pp. 695–708, 1987