A Metamodel Specification Tool Based on Description Logic and XML-DATA

Christophe Nicolle *

Abstract

Interoperation of heterogeneous and autonomous information systems has traditionally been hampered by semantic differences in their data models. With the World Wide Web, the development of cooperative information systems requires new solution to allow interoperation. In this paper, we address the problem by defining a methodology to help in the building of cooperative information systems. This metho-dology called X-TIME is based on a case-tool that allows to define automatically specific translators between coupled of heterogeneous data models composing the cooperation. This solution allows the dynamic extension of the cooperation to new heterogeneous information systems with a minimum work. It is best suited for World Wide Web environment where the number of heterogeneous databases that want to exchange data is very large.

Keywords : *Cooperation, extensibility, metamodel, translators, description logic, xml-data*

1 Introduction

The multiplication of information sources and the wild development of networks such as Internet requires new solution of interoperability. Interoperability is a new word that indicates a sharing of knowledge and expertise to allow the resolution of a common goal. In computer science, exchanging information between various information systems does interoperability. In database area, sharing data contained in DBMS makes interoperability.

1.1 Motivation

Data exchange between various heterogeneous databases is an intensive research area in the database world. Each year since 1983 [5], new architectures have been developed to merge data in a common representation. Distributed, Federated (loosely and tightly coupled), multi-base, and mediation architectures are defined to group legacy and remote databases in a cooperative information systems.
The corresponding methodologies try to combine three processes. The first process is a structural description of legacy information. This description is made through a model (global, pivot, meta, and other). Now, with new cooperative challenge such as the World Wide Web, extensibility has become the new main feature of these models. Extensibility is the feature of generic models, with generic concepts that can be specialised to map with the concepts of various data models. Unfortunately, a high level of abstraction is often associated with a low level of semantic. The second process is a semantic description. To keep

*LE2I - Université de Bourgogne. B.P. 47870, 21078 DIJON Cedex - FRANCE. email: cnicolle@u-bourgogne.fr

the meaning of initial data during the description process, old approaches advise to have confidence in the ability of the "cooperation administrator". The others propose to add constraints, meta-data, context or ontology to avoid or limit semantic losses. The third process concerns the mapping of both structural and semantic description from a representation (or data model) to another. This third process exists since the origin of data processing where it was necessary to explain to a computer what it had to make. Mapping is needed in the conception of information systems, the reengineering of information systems, the migration of information systems, and now the cooperation of information systems. In this last area, mapping becomes a very hard and heavy task due to the number of translators to define. Some approaches try to reduce the number of required translators by using a common representation or model. Thus, for n models the system needs n translators. For example, mediation architecture requires a Wrapper (a pseudo translator) for each cooperating information system. Building a translator requires a coding experience and a good knowledge of various techniques of data representation. In a wide area network such as World Wide Web, these requirements are insufficient due to the number of translators needed.

1.2 Scenario of Cooperation

To illustrate this paper, we describe an example of scenario for the cooperation of three heterogeneous databases in a loosely federated system. These databases (Relational, Codasyl and Object Oriented) cooperate by exchanging data. This is done, first, by exporting a part of each local schema to other databases, next, by querying importing schemas to get corresponding data from a source database. In our example, we focus on schema exchanges from the Relational to Codasyl and Object Oriented databases.

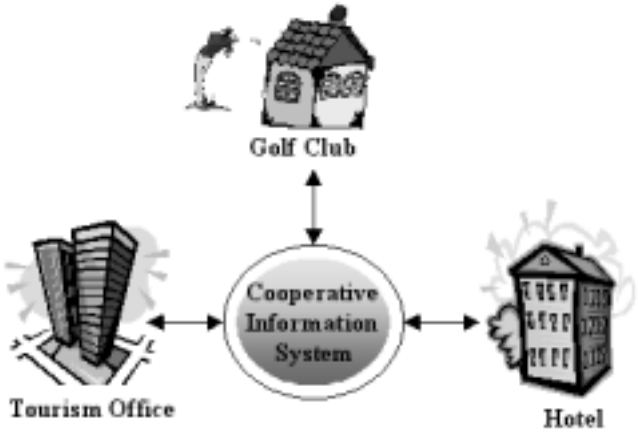

Figure 1: Cooperation of three heterogeneous information systems

The figure 1 depicts the cooperation scenario between a golf club using a Relational system, a hotel using an Object Oriented system and a tourism office using a Codasyl system. The goal of the cooperation is to complete activities allowed by the hotel and the tourism office with the various training possibilities of the golf club.
The relational schema describes the management of training courses, the members and competitions of the golf club. This schema depicts that a member can be registered for a competition if he has a sufficient level. A member can participate to a training course :

Periodic-Training (one day per week) or Week-Training (several grouped days on one week). A training course is organised by a Teacher and has a type (beginner, practice, ...). In the following, primary keys are noted with '#' and foreign keys are written in *italic*.

Competition (#Competition, name, date, min-level)
Member (#Member, name, first-name, level, address)
Registered (*#Competition, #Member*)
Participate (*#Member, #Training*)
Training (#Training, max-member, min-member, *type, teacher*)
Type (#Type, name, min-level)
Week-Training (*#Week-Training*, first-date, last-date)
Periodic-Training (*#Periodic-Training*, day, hours, time)
Teacher (#Teacher, name, first-name, level)

Inclusion dependencies are defined as follow :

Training.type $\subseteq$ Type.#Type
Training.teacher $\subseteq$ Teacher.#Teacher
Week-Training.#Week-Training $\subseteq$ Training.#Training
Periodic-Training.#Periodic-Training $\subseteq$ Training.#Training
Registered.#Member $\subseteq$ Member.#Member
Registered.#Competition $\subseteq$ Competition.#Competition
Participate.#Member $\subseteq$ Member.#Member
Participate.#Training $\subseteq$ Training.#Training

The export schema shared with other systems is composed of specific relations concerning training. These relations are : Training, Type, Week-Training, Periodic-Training. This schema is sent through the network to other databases. Each target database uses tools to translate this export schema into an import schema written in its own local model. In a loosely federated architecture, import schemas are locally integrated with the local schema. A user can access to federated data by querying this integrated local schema.
In this context, a hotel customer or a tourist can query the local integrated schema of Codasyl or OO database about proposed activities. His query (Codasyl or OO) requires access to data residing at both local and relational golf club database site. Thus, this query is decomposed into two sub-queries, one for local database and one sent by the network to the relational site. This last sub-query is translated into a relational query by translator tools to be handled by the relational site. Resulting relational data are sent back. Data received by the local site are formatted by a translator tool to be associated with local data (resulting from the first sub-query) to answer the initial query.

1.3 Related Work

To build a cooperative information system, a first step is collecting schemas of each local database. In a cooperative information system, all schemas exported from databases are translated and integrated into a cooperative schema which is used by the final user to query the cooperation "transparently" [15]. Several approaches have been proposed for the building of cooperative information systems.
These approaches can be classified in three categories: the approach using a unified global schema, the approach using a high-level representation language or a metamodel, and the approach using a direct mapping. The first approach consists in combining all information systems in a global system using integration techniques. This solution is adapted for tightly coupled federated systems that use a global schema to represent information exported by their component information systems. The schema is expressed in a global data model [17] in which the user of the federated system puts its requests. This method needs a translation at several levels.
First of all, a translation of data models as first integration process step, then a global request translation in local requests, and, finally, a data translation resulting from these requests.

Then, the local schema represents a view of the global schema. This solution needs a detailed analysis of all schemas and data. Therefore this is an expensive and intensive solution for a great number of database [11]. Moreover, this solution is not extensible to take into account the introduction of new systems.
The second approach using a metamodel allows the syntactic and/or semantic translation of component systems ([3, 4, 10]). This approach is very flexible and allows a great opening on others systems. Nevertheless, component systems have to express their requests in a high level language (this does not allow the access to Legacy database). Anyway, manipulation of high level languages is complex, and more, in metamodels, there is no unified interface on databases.
The third approach using direct mapping allows the resolution of heterogeneity by providing direct translation paths between each pair of data models composing the cooperative information system [2, 6, 7]. In this case, cooperative users can access to information through their own local interface. This solution is adapted to all cooperative systems and more particularly to loosely coupled federated systems described by Sheth in [17]. This approach becomes rapidly complex in term of number of translators required for an important number of heterogeneous data models. Indeed for n different models, it is necessary to build $n*(n-1)$ translators.
The above approaches propose generic models and frozen tools that are not extensible and not adaptable to various information systems. Nevertheless, they have pointed out the need to reduce the number of translators requires to allow interoperability of information systems. Tools that are based on the reuse of pre-defined concepts and translations can be used to make the generation of one to one data model translators easier [13, 18]. The semiautomatic or automatic generation of one to one translators thus facilitates the translation in cooperative information systems. To reach this goal, we propose a case tool called X-TIME, which helps in the definition of schema translators for each cooperating information systems. This case tool is based on an extensible metamodel coupled with transformation rules. In this paper we focus on the main step of our solution : The building of a metamodel and tools corresponding to specific cooperating information systems. The remainder of this paper is organised as follows : Section 2 presents the requirements of our methodology. Section 3 describes the metamodel building step on the example presented in section 1.2. Section 4 presents an overview of our translation methodology. The last section concludes this paper.

2 Requirements

Existing solutions propose a static model (super model, canonical model or meta-model) defined from a limited set of data models [3, 4, 10]. These solutions best suited to the integration of homogeneous databases (relational) are not well adapted to wide networks such as internet where a great variety of information systems co-exist. Cooperation of various information systems requires an extensible model. A first solution has been proposed by Barsalou in [4] with a Metamodel where meta-concepts (or metatypes) are organised in an inheritance lattice. Unfortunately, the addition of new concepts only took place by the specialisation of existing meta-concepts without reorganisation of the metamodel structure. Thus, as shown in figure 2 the metamodel structure depended on the order in which information systems were integrated in the cooperation.
To solve this problem, it is necessary to proceed to a classification step before adding concepts in the metamodel. This classification, automatically achieved, guarantees the reorganisation of the metamodel to every new addition of an information system in the cooperation. This classification is done in three parts. First of all concepts of data models are formally

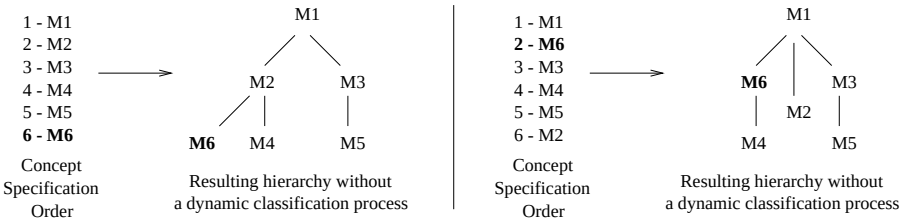

Figure 2: Without Subsumption, order is required in concept specification

described using description logic. Then, a mechanism of subsumption is going to analyse the formal definition of these concepts to organise them in a hierarchy of specialisation (see figure 3).

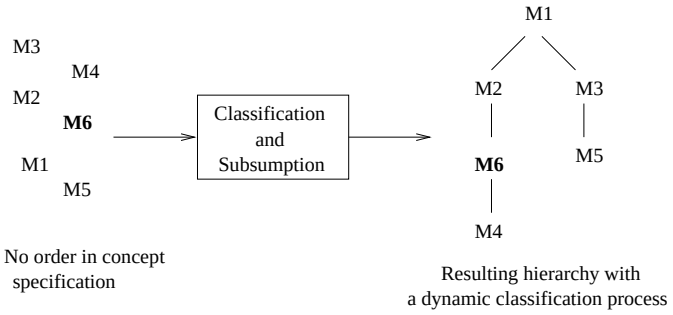

Figure 3: Classification of concepts and subsumption mechanism

Finally, the formal definition of every concept is simplified according to the place they occupy in the hierarchy (thanks to the inheritance mechanism). The formal definition of concepts is made only one time, independently of the number of cooperating information systems. The subsumption and the simplification of the definition concepts are made automatically each time a new system is introduced in the cooperation. This guarantees the building of a specific metamodel. To complete the metamodel definition, we associate to each metatype a syntactic definition in XML-DATA. We will see that XML-DATA replaces advantageously the usual BNF definitions.
In this section, we first present some features of Description Logic and XML-DATA. Then we present the generic metatype, which is defined to model all possible properties of metatypes, which represent data models features.

2.1 Main Features of Description Logic

This part briefly presents main concepts on description logic issued from the KL-ONE family. KL-ONE [22] is a system allowing knowledge representation and automatic classification using the subsumption mechanism. This presentation focuses on four basic notions of description logic used to model and classify knowledge in the Strategic Hierarchy Builder : roles, concepts, individual concepts and subsumption mechanism.

Role. It is an oriented relation from an owner concept to a member concept. Cardinalities can be attached on a role. These cardinalities represent minimum and maximum numbers of

possible instances of the member concept linked with the owner concept. A role is graphically defined by an arrow with a symbol (circle and/or square grouped). A role can be specialised (dotted arrow in figure 4 between *r2* and *r1*) to build a new role with the same structure but with new constraints. Using the syntax of the system Back [9], which is a specific implementation of description logic, the description of a role is realised by specifying the owner concept as "domain" and the member concept as "range".

Concepts : A concept can be viewed as a real world entity. The predefined concept "anything" which is the most general concept subsumes all others. A concept inherits roles from the concepts by which it is subsumed. The specialisation of a concept can be directly realised giving father concepts in the hierarchy. It inherits attached roles from father concept (dotted arrow in figure 4 between *r1* and *r2*).

Individual concepts : A distinction is made between generic and individual concepts corresponding roughly to the difference between classes and their instances. To simplify description, individual concepts can be viewed as real world objects.

Subsumption mechanism : A subsumption mechanism classifies defined concepts. A concept named A subsumes a concept named B, if the definition of A is more general than the definition B. Using the basic mechanism and applying tests on all defined concepts, an inheritance hierarchy of concepts can be found. This mechanism is a key point of the Strategic Hierarchy Builder described in this paper.

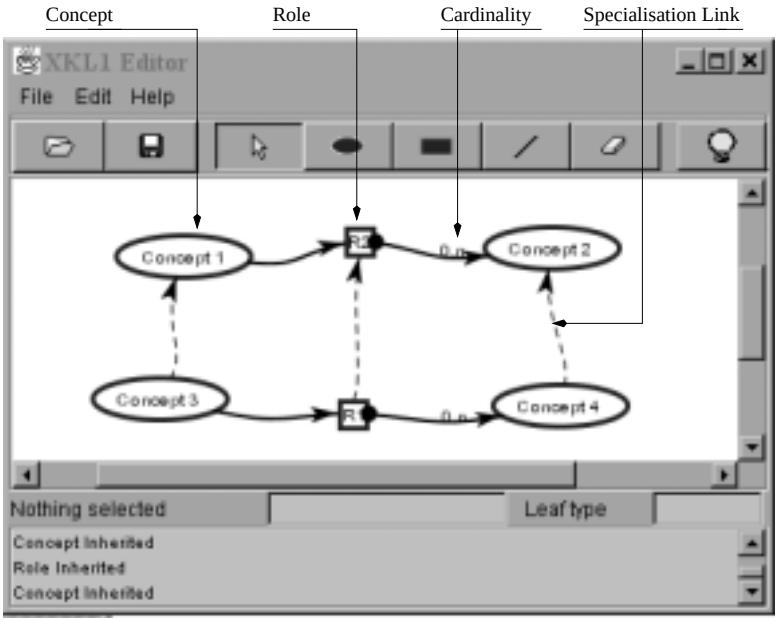

Figure 4: Example of description logic using XKL1Editor

2.2 Main Features of XML-DATA

XML-Data is a meta-language used to create schemas, identify structures and constraints in an XML document. A XML-DATA schema is considered as a well formed XML document. In this article, we use XML-DATA to model the structure and the constraints of metatypes.

This choice is based on many motivations. First, XML-DATA allows us to define extensible schema. This extensibility is achieved by specialising existing elements or existing attributes to add new elements or attributes. This feature is well adapted in our methodology. Next, this language has the property to support the inclusion of many documents as parts of one document. In database area, this property is derived to integrate many schemas in one cooperative schema. Last, XML-DATA allows to web applications to exchange data using a standard coding. Thus this feature will allow to exchange data between heterogeneous information systems using intermediate standard representation which can be handled by any browsers or XML tools. Detail descriptions of XML and XML-Data can be found in [1, 21, 20, 16].

2.3 Basic Metatype Definition

There are two sorts of metatypes: specific and basic metatypes. Specific metatypes are used to capture the semantics of different categories of modelling concepts. In our example, the cooperation is built from the specification of three sets of metatypes. The first set contains concepts of the Tourism Office model (*record* and *set* concepts of the Codasyl data model). The second set contains concepts of the Hotel model (*Class* and *Inheritance Link* concepts of the OO data model). The third set contains concepts of the Golf Club model (*Table*, *Primary Key* and *Foreign Key* concepts of the relational data model). Basic metatypes, which initially composed the metamodel, are defined to help in the definition of specific metatypes. Initially they are defined from a generic concept call "metatype" of which the definition is given in appendix (section 6). A metatype M is defined by a tuple $M = (A_M, C_M, P_M)$, where A_M is a set of syntactic elements that describe the structure of M. C_M is a set of user defined constraints that are used to restrict the meta data constraints associated with the super metatype of which M is a specialisation. P_M is a set of operations or methods. It is used to model methods of the object oriented (or any similar) model. P_M is empty for data models (e.g. relational model) which do not allow the encapsulation of data and operations into a type.
In the following, we describe the basic metatypes. Section 3 describes the set of specific metatypes corresponding to our example.

- Metatype META

  The highest metatype in the specialisation hierarchy is a generic metatype, META, with an empty structure. Its purpose is to define meta data constraints and operations that can be shared by all metatypes. It is defined by META=([ ], C_{Meta}, [ ]). C_{Meta} comprises a set of data modelling constraints. For example, C_{Meta} contains an ID function that uniquely identifies the instances of metatypes. The description logic of META is given in table 1.

- Object Metatypes

  Metatype Complex-Object represents modelling concepts that are used to describe complex structure entities such as: CLASS in the object oriented model, ENTITY TYPE in the Entity-Relationship model and RECORD in the Codasyl data model. Metatype MComplex-Object is specified by CO = (A_{CO}, C_{CO}, P_{CO}), where A_{CO} is defined using the usual tuple and set constructions on meta object types. The component C_{CO} and P_{CO} are not redefined at this level, but are inherited from the super metatype META.

  Metatype MSimple-Object, which is a specialisation of metatype MComplex-Object, represents modelling constructs with flat structure. It is defined by SO = (A_{SO}, C_{SO},

```
'META':= 'metatype'
     and atmost(1,r_description)
     and atmost(1,r_link)
     and atmost(1,r_attribute)

'MComplex-Object' := 'metatype'
     and exactly(1,r_description)
     and exactly(1,r_description_label)
     and exactly(1,r_identifying_description)
     and no(1,r_link)
     and atleast(1,r_attribute)
     and exactly(1,r_attribute_label)
     and exactly(1,r_attribute_type)
     and exactly(1,r_attribute_cardinality)

'MNary-Link' := 'metatype'
     and exactly(1,r_description)
     and exactly(1,r_description_label)
     and exactly(1,r_identifying_description)
     and all(r_identifying_description,'OID')
     and exactly(1,r_link)
     and atmost(1,r_attribute)
     and exactly(1,r_attribute_label)
     and exactly(1,r_attribute_type)
     and exactly(1,r_attribute_cardinality)
```

```
'MSimple-Object' := 'metatype'
     and exactly(1,r_description)
     and exactly(1,r_description_label)
     and exactly(1,r_identifying_description)
     and no(1,r_link)
     and atleast(1,r_attribute)
     and exactly(1,r_attribute_label)
     and exactly(1,r_attribute_type)
     and exactly(1,r_attribute_cardinality)
     and all(r_attribute_type, simple_type_attribute)
'MBinary-Link' := 'metatype'
     and exactly(1,r_description)
     and exactly(1,r_description_label)
     and exactly(1,r_identifying_description)
     and all(r_identifying_description,'OID')
     and exactly(1,r_link)
     and all(r_link,'binary')
     and atmost(1,r_attribute)
     and exactly(1,r_attribute_label)
     and exactly(1,r_attribute_type)
     and exactly(1,r_attribute_cardinality) //
```

Table 1: Basic Metatypes Description

P_{SO}) where the attribute structure A_{SO} inherits the tuple structure A_{CO} of metatype MComplex-Object, but restricts the type of its components to primitive domains. The components C_{SO} and P_{SO} are inherited from the metatype MComplex-Object.

- Link Metatypes

  Metatype MNary-Link models types that represent connections between real world entities. Links can carry attributes. It is defined by NL = (A_{NL}, C_{NL}, P_{NL}). The component C_{NL} and P_{NL} are not redefined at this level, but are inherited from the super metatype META.

  Metatype BL categorises binary connections involving real world object types. It is a special case of MNary-Link and it is defined by BL = (A_{BL}, C_{BL}, P_{BL}). In the following we do not give the XML-DATA of MNary-Link and MBinary-Link which are directly inherited from the type elements "n_ary" and "binary" (see appendix in section 6).

The use of the sumsumption mechanism on the basic metatypes gives a hierarchy depicts in figure 5.

3 Specific Metamodel Building

To construct a cooperation, every information system must describe data model that it uses. For it, every site uses a tool called XKL1Editor that allows to describe concepts of the local data model at semantic and syntactic level. The semantic part is described with the help of description logic [22], the syntactic part is described with the help of the XML-Data meta-language [20]. The semantic description uses logical terms to describe concepts. Every

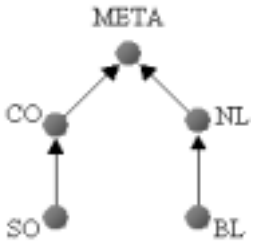

Figure 5: Resulting hierarchy using subsumption on the set of basic metatypes

clause generates a question where the local administrator must give a syntactic description (XML-Data) of the element. The semantic description always begins by a specialisation of a generic metatype that owns predefined clauses. Once the process of each data model description is done, it is necessary to regroup information of every site to launch the process of construction of the corresponding metamodel.
Although the metamodel is initially composed of a set of basic metatypes, its final composition depends on data models that compose cooperation. The metamodel is dynamically constructed by a mechanism of subsumption. This mechanism compares the semantic definition (description logic) of each element described by local database administrators. This comparison is automated in a tool called Strategic Hierarchy Builder. The dynamic construction of the metamodel permits to have always a metamodel adapted to information systems that want to co-operate. Moreover, this solution simplifies the addition of new information systems in the cooperation. The result of this construction is an a-cyclic graph where each node is a metatype corresponding to one or several elements described by local administrators and where each arc represents a specialisation link. This organisation allows the reuse of the definition of every metatypes and then to simplifiy its definition to make only appear differences between metatypes. At this level, we have obtained a specific metamodel that describes the syntax and the semantic of each concept used in the cooperating information systems. Figure 6 depicts our example of specific metamodel. This metamodel contains five basic metatypes (META, CO, SO, NL, BL) and five specific metatypes (REL for relational data model, CLA and IHL for object oriented data model, REC and SET for Codasyl data model). Arrows model specialisation links.

3.1 Specification of the Golf Club Data Model Concepts

The data model of the Golf club is a relational data models. The result of the specification step of relational concepts is a metatype MRelation (REL). The meta-type MRelation represents the concept RELATION (relation name, mono-valued attributes, on simple domains, functional and inclusion dependencies). It is defined by REL=(A_{REL}, C_{REL}, P_{REL}). The structure A_{REL} represents objects with flat structure (mono-valued, simple. C_{REL} models the relational primary key; namely, a sub-sequence of attribute labels that uniquely identify the tuples of a relation. Moreover, C_{REL} models foreign key. P_{REL} is empty. Table 2 and figure 7 gives the description logic and XML-DATA definition of MRelation.

3.2 Specification of the Tourism Office Data Model Concepts

The data model of the Tourism Office is a Codasyl data model. The result of the specification step of the Codasyl concepts is a metatype MRecord (REC) and a metatype MSet (SET).

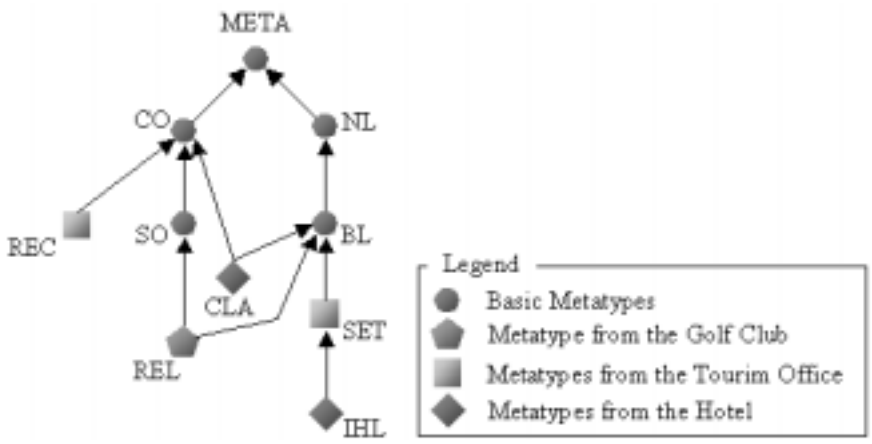

Figure 6: The resulting specific metamodel

The meta-type MRecord specialises meta-type MComplex-Object. It is defined by REC = (A_{REC}, C_{REC}, P_{REC}). The component A_{REC} is directly defined from A_{CO} The components P_{REC} and C_{REC} are empty. The meta-type MSet is an oriented binary link with an owner and a member. The meta-type MSet is defined by SET = (A_{SET}, [ ], C_{SET}). C_{SET} contains specific constraints of SET concept.

3.3 Specification of the Hotel Data Model Concepts

The data model of the Hotel is an object oriented data model. The result of the specification step of the object oriented concepts is a metatype MClass (CLA) and a metatype MInheritance-Link (IHL).
As illustrated in figure 6, the meta-type MClass specialises meta-type MComplex-Object and MBinary-Link. It is defined by CLA = (A_{CLA}, C_{CLA}, P_{CLA}). The structure A_{CLA} is a mixed between the structure of A_{CO} and the structure of A_{BL}. The inheritance from A_{BL} allows the representation of reference attributes. Conversely, The meta-type MClasse refines the component P_{CO} to introduce the behavioural aspect of the objects. A detailed and formal description of methods is beyond the scope of this paper. Thus the component P_{CLA} is not presented. C_{CLA} is empty. The corresponding XML-DATA definition is presented in figure 8.
The meta-type MInheritance-Link is a binary link. In addition to the constraints inherited

```
'MRelation_A' := 'metatype'
        and exactly(1,r_description)
        and exactly(1,r_description_label)
        and exactly(1,r_identifying_description)
        and all(r_identifying_description,'Primary_key')
        and atleast(1,r_attribute)
        and exactly(1,r_attribute_label)
        and exactly(1,r_attribute_type)
        and exactly(1,r_attribute_cardinality)
        and all(r_attribute_type,simple_type_attribute)
```

```
'MRelation_C' := 'metatype'
        and exactly(1,r_link)
        and all(r_link,'binary')
        and all(r_link,'foreign_key')
```

Table 2: MRelation Description in DL

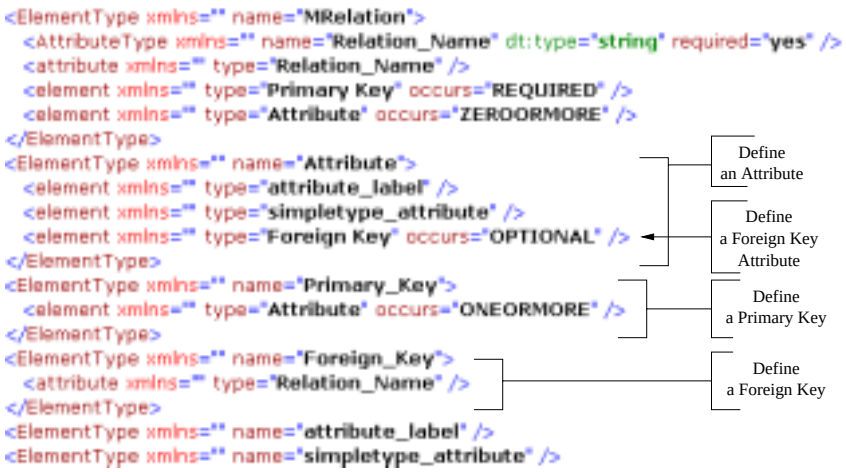

Figure 7: MRelation Description in XLML-DATA

from the meta-type MSet, it maintains a constraint of subset between the population of the specialised and generalised meta-types which generalises the concept of inheritance of the object oriented data model. Moreover, MInheritance-Link specialises the structure of MBinary-Link to represent connection without attribute. The meta-type MInheritance-Link is defined by IHL = (A_{IHL}, C_{IHL}, [ ]).

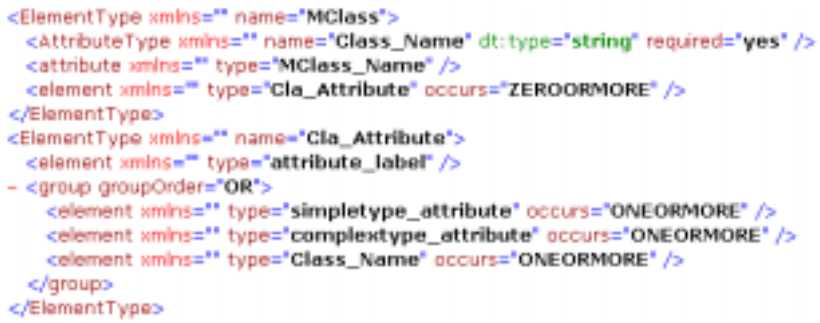

Figure 8: MClass description in XML-DATA

4 Translators Building

To complete our methodology, we have developed tools to generate translators using the metamodel hierarchy. These translators will allow to exchange schemas between heterogeneous sites. To build these translators, we associate a transformation rule between certain metatypes. At this level, we will see that the number and the difficulty of transformations are widely reduced. When transformation rules are defined, a translator compiler automatically builds all translators needed to exchange heterogeneous schemas in the cooperation.

4.1 Transformation Rules Generation Step

The hierarchy of specialisation defined in the previous step allows the reduction of the number of transformation rules needed to build the translators. Rather than defining rules between all metatypes, it is sufficient to define rules between every couple of metatypes joined by an arc (figure 9). The number of rules to define by couple of metatype depends of several factors:

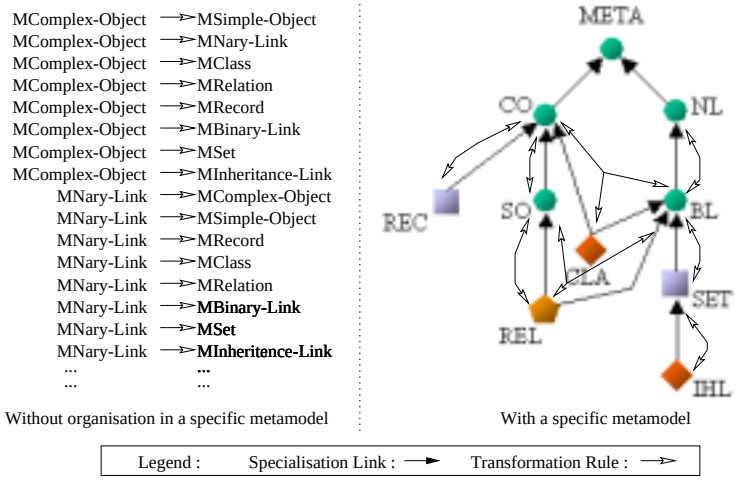

Figure 9: Comparison with / without metamodel

- META is not considered like a metatype that can be translated. It provides generic notions that are specialised by the other metatypes. There is not rule for arcs joining META to other metatypes. For example in figure 9, the number of efficient metatypes to translate is 9 (All except META). The number of arcs is 9 (all except those join to META).

- For metatypes that only have one father, there is one rule from the son to its father and one rule from the father to its son. Therefore generally, an arc needs two transformation rules. On figure 9, this case is applied to 7 metatypes (except CLA and REL). There are 14 rules to define.

- For metatypes that have several fathers, there is one rule from the son to its fathers. This rule assures the transformation of a son metatype instance into one or several father metatype instances. Inversely, there is one rule by arc joining fathers to a common son. For example, in figure 9, the metatype CLA possesses two fathers (CO and BL) as well as REL (SO, BL). There are three rules defined for each of these two metatypes: First, A rule from CLA to the couple CO and BL; Next, a rule from CO to CLA; Last, a rule to BL to CLA. It is the same way for REL (REL to BL and SO, SO to REL and BL to REL).

Thus, the number of rules does not depend on the number of metatypes but depends on the number of arcs. For example, in figure 9, instead of defining 72 transformation rules for 9 metatypes (9 metatypes * (9-1) metatypes), it is sufficient to define 16 transformation rules (5 simple arcs * 2 (sense of rules) + 3 multiple arcs * 2 metatypes).

Transformation rules are used to convert an XML-DATA schema modelling a source metatype into another XML-DATA schema modelling a target metatype. To build transformation rules, we have developed in Java a tool called Translation Rule Builder (TRB). This tool provides a set of functions, which helps in the definition of rules that manipulate metatypes. These functions are simple functions such as change, replace, delete, create, insert, and find a string in a file, and complex functions such as mathematical and logical functions. Moreover, some rules can be built by the combination of other rules. At the end, we have a collection of pre-defined rules that can be combined to built a specific transformation rule. Rules are stored in a Transformation Rule Library. Thus each rule defines with the TRB is automatically generated in JAVA. The creation of new rules by combination of existing rule generates a corresponding Java code.

4.2 Translator Construction Step

When the metamodel and rules are specified, it is necessary to build translators. This operation takes place in three steps with the help of a tool called " Translator Compiler " [12].
The first step determines transformation paths. A transformation path joins two distant metatypes in the metamodel. In our example, the transformation path between MRelation and MComplex-Object is [REL, SO, CO] (see figure 10). The construction of transformation paths is achieved using Dijkstra's algorithm [8].
The second step consists in regrouping transformation paths to create a translation path. This step is achieved by successive regrouping of transformation path.

- First of all, transformation paths that possess the same source metatype and the same target metatype are regrouped. In our example the path between MRelation and MClass is constructed from two transformation paths [REL, SO, CO, CLA] and [REL, BL, CLA]. This step allows to sole problems due to multiple inheritance. The resulting path is [REL, [SO, BL], [CO, BL], CLA] (see figure 10).

- Next, paths between metatypes generalising a source model and metatypes generalising a target model are regrouped. In figure 10, the translation path between a relational model (REL) and an object model (CLA, IHL) is [REL, [SO, BL], [CO, BL], [CLA, BL], [CLA, SET], [CLA, IHL]].

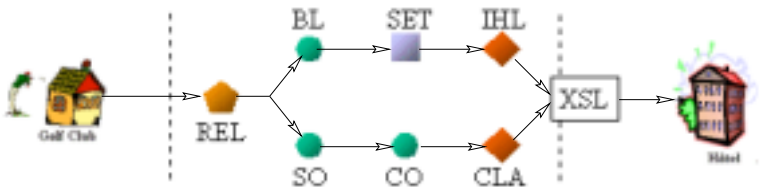

Figure 10: Transformation paths from relational to object and Codasyl metatypes

The last step is the compilation of transformation rules according to the resulting translation path. This compilation generates a specific translator that allows the translation of a schema from a source to a target data model and inversely. An association of the code of every rule is done to obtain this translator. Then an optimisation of this code is made. The Translator

Compiler provides the java source code of the translator. Therefore, it is possible to build the corresponding executable file on the various platforms composing the cooperation. To complete the definition of the resulting translators we associate XSL stylesheet [1, 19] to each cooperating systems. This stylesheet are used to represent the final translated XML-DATA schema in the target database syntax. XSL is an XML eXtensible Stylesheet Language initially defined to write transformations from XML to HTML. However it can also serve in data applications.

4.3 Metatype Definition Simplification

When organised, cooperation can be spreat to new heterogeneous information systems. It is sufficient for these new systems to specify concepts of their data model with the help of the XKL1Editor tool. The resulting metatypes are integrated to the list of metatypes existing in the cooperation. The Strategic Hierarchy Builder tool can rebuild a new metamodel adapted to this new cooperation [14]. The number and the type of transformation rules to define depend on the resulting hierarchy. It exists several possibilities.

- If a new metatype (New-M) is connected to the hierarchy of specialisation as a leaf of an unique father (M2), then it is necessary to define two transformation rules (one from the father to the son, one from the son to the father, figure 11.(a)).
- If this metatype (New-M) leaf possesses several fathers (M1, M2), it is necessary to define a rule from this metatypes to its fathers and a rule from each father to the son (figure 11.(b)).

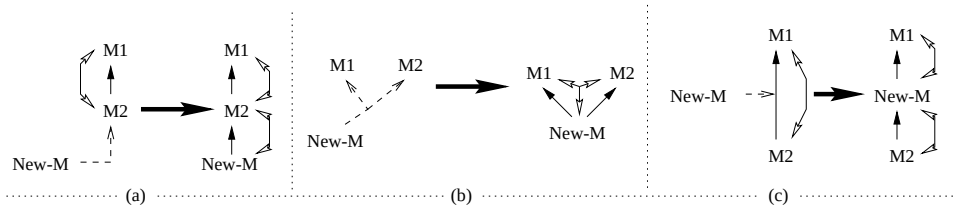

Figure 11: Addition of a new metatype in the hierarchy

- If this new metatype (New-M) possesses sons (M2). It is necessary to define as many rules as sons. In this case, pre-existing rules can be decomposed and reused (figure 11.(c)).

In our example, the resulting hierarchy is presented in figure 6 where REL inherits from SO and BL metatypes. Thus, three transformation rules must be defined: One to transform REL instances into SO and BL instances, one to transform SO instances into REL instances and one to transform BL instances into REL instances. This number of rules is independent of the cooperation size and the number of heterogeneous data models cooperating. Moreover, the definition of all metatypes is simplified. For example, table 3 shows the new definition of MSimple-Object (SO) and MRelation (REL) metatypes.

5 Conclusion

In this paper we have presented a methodology for translating multiple data models. We address the problem by defining tools that allow the construction of specifics metamodels. The

'MRelation' := 'MSimple-Object' and 'MBinary-Link' and all(r_identifying_description,'Primary_key') and all(r_link,'foreign_key')	'MSimple-Object" := 'MComplex-Object' and all(r_attribute_type, simple_type_attribute)

Table 3: New MSimple-Object and MRelation Description

dynamic construction of the metamodel allows having a metamodel adapted to information systems that want to make part of the cooperation. Moreover, it makes the addition of a new information system in the cooperation easier. The addition of a new model in the cooperation is done locally by a description of local data model concepts using XKL1Editor. The resulting definitions in description logic and XML-DATA allow the construction to construct a new metamodel adapted to the new cooperation as well as specific translators. The specialisation hierarchy allows first to reuse the definition of every metatype by a mechanism of inheritance and especially to simplify their definition. The X-TIME method permits to create extensible and dynamic cooperations. The main characteristics of resulting metamodels are:

- It provides a minimum set of metatypes that capture the semantics of different categories of concepts found in data models composing the cooperation.
- It achieves extensibility by organising the metatypes in an specialisation hierarchy. Thus, a new metatype is defined by specialising an existing metatype.
- It achieves translation by defining a set of transformation rules translation paths and translators.
- It allows the reuse of transformation rules and sharing of translation step to reduce the work of translators building.

Our future objectives are to extend the above results, and to define a formal methodology and algorithm for heterogeneous query processing. This will allow us to define a query interface for the interoperation or migration of existing systems.

6 Appendix

In this section, we present the description of the generic metatype. A generic metatype description is defined to model all possible properties of metatypes, which represent data models features. These properties can be the ability for the metatype to possess a label, to manipulate concepts such as identifying concepts (keys, OID, etc.), to be described by attributes (flat structures, complex structures, etc.), to manage link concepts (relation, binary relations, N-ary relations, etc.). To model the generic metatype, a set of description logic concepts are defined (Figure 12). An XML-DATA definition of the generic metatype is given also. This definition is reused to help in the XML-DATA definition of new metatypes. For each specified clause, we give the corresponding XML-DATA definition.

- *the description concept.* This concept is introduced to model the ability for a data model described by metatypes to manage identifying concepts such as label, key or OID. In description logic the notion of *description*, *description label* and *identifying description* are three concepts. Roles link the concept of *description* with others. The formal description is given by:

Figure 12: Description Logic of Generic Metatype

```
description :< anything
label_description :< anything
identifying\_description :< anything
rc_label_description :< domain(description) and range(label_description)
rc_identifying\_description :< domain(description)  and range(identifying\_description)
```

The corresponding XML-DATA definition of the description concept is:

```
<elementType id="description_label">
    <string/>
</elementType>

<elementType id="description_identifier">
    <key id= "oid"><KeyPart href="#description_identifier"/></key>
</elementType>

<elementType id="description">
    <element type="#description_identifier" occurs="REQUIRED"/>
    <element type="#description_label" occurs="REQUIRED"/>
</elementType>
```

- *the attribute concept* is introduced to specify which kind of attribute can be managed by data models. It is defined by three components: a type of structure (simple with flat structure, complex with lists, sets, etc.), an attribute label and which kind of cardinalities can be used with the structure (mono valued attributes, multi-valued attributes). In description logic the notion of *attribute*, *label attribute*, *type attribute* and *cardinality attribute* are defined as concepts. The last two concepts are specialised to introduce notions of simple and complex attributes and notions of mono-valued and multi-valued cardinalities. The *attribute* concept is linked by a role to others.

```
attribute :< anything
label_attribute :< anything
type_attribute :< anything
simple_type_attribute :< type_attribute
complex_type_attribute :< type_attribute
cardinality_attribute :< anything
mono_valued_attribute :< cardinality_attribute
multi_valued_attribute :< cardinality_attribute
rc_label_attribute :< domain(attribute) and (label_attribute)
rc_type_attribute :< domain(attribute) and range(type_attribute)
rc_cardinality\_attribute :< domain(attribute) and range(cardinality\_attribute)
```

The corresponding XML-DATA definition of the attribute concept is:

```
<elementType id="simpletype_attribute ">
    <superType type="#attribute_type"/>
</elementType>

<elementType id="complextype_attribute ">
    <superType type="#attribute_type"/>
    <group groupOrder="OR">
       <instance_name>string/</instance_name>
       <element type="#attribute" occurs = REQUIRED"/>
    </group>
</elementType>

<elementType id="attribute_label ">
    <string/>
</elementType>

<elementType id="attribute_type ">
    <string/>
</elementType>

<elementType id="cardinality ">
    </string>
</elementType>

<elementType id="attribute_cardinality ">
    <element type="#cardinality" occurs="REQUIRED"/>
    <element type="#cardinality" occurs="REQUIRED"/>
</elementType>

<elementType id="attribute">
    <element type="#attribute_type" occurs="REQUIRED"/>
    <element type="#attribute_label" occurs="REQUIRED"/>
    <element type="#attribute_cardinality" occurs="REQUIRED"/>
</elementType>
```

- *the link concept* which allows to represent the general notion of data model relationship (binary or N-ary relationship, inheritance relationship, etc.) In description logic, the notion of *link* is defined as a concept and is specialised to introduce the binary and N-ary link concepts.

```
link :< anything
binary :< link
nary :< link
```

The corresponding XML-DATA definition of the link concept is :

```
<elementType id="binary ">
    <superType type="#link "/>
    <element type="#concept-name" occurs="REQUIRED"/>
    <element type="#concept-name" occurs="REQUIRED"/>
</elementType>

<elementType id="n_ary ">
```

```
    <superType type="#link"/>
</elementType>

<elementType id="Concept-name ">
    </string>
</elementType>

<elementType id="link">
    <element type="#concept-name" occurs="REQUIRED"/>
    <element type="#concept-name" occurs="ONEORMORE"/>
</elementType>
```

At this step of specification all basic concepts to specify the generic notion of *metatype* are defined in description logic. The concept *metatype* is linked by roles to *description*, *attribute* and *link* concepts defined above.
the formal definition is :

```
r_description :< domain(metatype) and range(description)
r_link :< domain(metatype) and range(link)
r_attribute :< domain(metatype)  and range(attribute)
```

Composed roles are created to link 'metatype' to other concepts. To simplify figure 12 composed roles are not drawn.

```
r_label\_description := r_description comp  rc_label_description
r_identifying_description := r_description comp rc_identifying_description
r_label_attribute := r_attribute  comp rc_label_attribute
r_type_attribute := r_attribute  comp rc_type_attribute
r_cardinality_attribute := r_attribute comp rc_cardinality_attribute
```

The corresponding XML-DATA definition to model the generic metatype is :

```
<elementType id="metatype">
  <instance_name><string/></instance_name>
  <element type="#attribute" occurs="ZEROORMORE"/>
  <element type="#description" occurs="ZEROORMORE"/>
  <element type="#link" occurs="ZEROORMORE"/>
</elementType>
```

References

[1] S Abiteboul, P Buneman, and D Suciu. Data on the web. In *From Relations to Semistructured Data and XML*, http://mkp.com, 2000. Morgan Kaufmann Publishers.

[2] M Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. *Proceedings of the 13th International Conference of Entity-Relationship Approach*, pages 403–419, december 1994. Manchester UK.

[3] P Atzeni and R Torlone. Mdm : A multiple-data-model tool for the management of heterogeneous database schemes. *Proceedings of the SIGMOD International Conference*, pages 538–531, 1997.

[4] T Barsalou and D Gangopadhyay. M(dm) : An open framework for interoperation of multimodel multidatabasesystems. *Proceedings of the 8th International Conference of Data Engineering*, pages 218 – 227, Febuary 1992. Tempe, Arizona.

[5] C Batini and M Lenzerni. A methodology for data schema integration in the entity-relationship model. *Proceedings of the 3rd International Conference on Entity-Relationship Approach*, pages 413–420, 1983. North Holland.

[6] M Blaha, W Premerlani, and H Shen. Converting oo models into rdbms schema. *IEEE Transaction*, pages 28–39, 1994.

[7] S Cluet, C Delobel, J Simon, and K Smaga. Your mediator need data conversion! *ACM SIGMOD International Conference on Management of Data*, pages 177–188, June 1998.

[8] O J Dahl, E W Dijkstra, and C A R Hoare. Structured programming, second printing. *A.P.E.C. Studies in Data Processing No.8, Academic Press*, 1973. London-New-York.

[9] T Hoppe, C Kindermann, J J Quantz, A Schmiedel, and M Fischer. Back v5, tutorial and manual. *Technische Universitat Berlin, Projekt KIT-BACK*, March 1993. Berlin, Germany.

[10] M A Jeusfeld and U A Johnen. An executable metamodel for re-engineering of database schemas. *Proceedings in the 13th International Conference of Entity-Relationship Approach*, 1994. Manchester, UK.

[11] L V S Lakshmanan, F Sadri, and I N Subramian. On the logical foundations of schema integration and evolution inheterogeneous database systems. *Deductive and Object-Oriented Database, 3rd International Conference, LNCS 760*, 1993. Phoenix, Arizona.

[12] C Nicolle. A translator compiler for interoperable information systems. *17th International CODATA Conference*, October 2000. Baveno, Italy.

[13] C Nicolle, D Benslimane, and K Ytongnon. Multi-data models translation in interoperable information systems. In Springler Verlag, editor, *Lecture Notes in Computer Science*, pages 78–89. CAISE, May 1996.

[14] C Nicolle, N Cullot, and K Yetongnon. Shb : A strategic hierarchy builder for managing heterogeneous databases. *International Engineering and Applications Symposium*, August 1999. Montreal, Canada.

[15] M Papazoglou, N Russel, and D Edmond. A translation protocol acheiving consensus of semantics between cooperating heterogeneous database systems. *Proceeding of the First IFCIS International Conference on Cooperative Information Systems*, pages 78–89, june 1996. Brussels, Belgium.

[16] W J Pardi. Xml. In *in Action*, http://www.microsoft.com/france/mspress, 1999. Microsoft Press.

[17] A P Sheth and J A Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22, 1990.

[18] S Spaccapietra, M Andersson, Y Dupont, K Ytongnon, C Parent, and C Nicolle. Integrating schemas of heterogeneous database systems. *Proceeding of the Second Meeting on the Interconnection of Molecular Biology Databases (MIMBD95)*, July 20-22 1995. Cambridge UK.

[19] W3C. Extensible stylesheet language (xsl). *http://www.w3.org/XSL/*, 1998.

[20] W3C. Xml-data, w3c note 05 jan 1998. *http://www.w3.org/TR/1998/NOTE-XML-data*, January 1998.

[21] W3C. Xml web page. *http://www.w3.org/XML/*, 1998.

[22] W A Woods and J G Schmolze. The kl-one family. *Computer Mathematic Application*, 23:133–177, 1992.