XML integration and toolkit for B2B applications

Christophe Nicolle, Kokou Yétoignon and Jean-Claude Simon

Laboratoire LE2I, Université de Bourgogne, BP 47870, Dijon Cedex – France

{cnicolle, kokou, jean-claude.simon}@u-bourgogne.fr
Abstract

This paper presents a Web based data integration methodology and tool framework, called X-TIME, for the development of Business-to-Business (B2B) design environments and applications. X-TIME provides a data model translator toolkit based on an extensible metamodel and XML. It allows the creation of adaptable semantics oriented metamodels to facilitate the design of wrappers or reconciliators (mediators) by taking into account several characteristics of interoperable information systems such as extensibility and composability. X-TIME defines a set of meta-types for representing meta-level semantic descriptors of data models found in the Web. The meta-types are organized in a generalization hierarchy to capture semantic similarities among modeling concepts of interoperable systems. We show how to use the X-TIME methodology to build cooperative environments for B2B platforms involving the integration of Web data and services.
XML integration and toolkit for B2B applications

B2B applications based on business information systems interoperability are increasingly available on the Internet. The Gartner Group (InfoWorld, 2000) estimates that B2B (B2B, 2001) revenue worldwide will reach $7.29 trillion dollars by 2004. These emerging systems involve the exchange of both data and services among business information systems. They have created many challenges, including the need for novel interoperability techniques and architecture. Emerging B2B integration approaches must answer several questions: 1) how to process and share data in various business formats? And 2) how to integrate various business functionalities into Web services? Interoperability is generally hampered by heterogeneity issues. Platform (hardware, software, communication) heterogeneity is resolved by communication standards and protocols such CORBA, IP and HTTP. Syntactic heterogeneity requires common or pivot metamodels to represent the data of the participating systems. Finally semantic heterogeneity, which is difficult to tackle, requires semantic models and languages that can capture the meaning of the representation concepts of different information systems.

In addition to traditional data integration concerns, web-service integration must allow the exchange of business services and processes by standardizing 1) low level communication systems and protocols (SOAP: Simple Object Access Protocol), 2) data presentation format and definition languages (WSDL: Web Services Description Language), and 3) access and classification of Web services (UDDI: Universal Discovery, Description and Integration). WSDL is a web-service definition language aimed at the resolution of structural data heterogeneity while UDDI (UDDI, 2002) provides a library of Web services and their specifications.

XML is increasingly used in the development of Web services and B2B integration and is emerging as a de facto standard for data exchange in networked environments (Abiteboul S.,
Buneman P., Suciu D., 2000; XML, 2000). XML (eXtensible Markup Language) is an open
textual language that provides a structural information description and relative semantics to data
(Pardi W. J., 1999). XML is more than a tool or language for separating content from
presentation: it is a meta-language from which more than 300 languages (ZapThink, 2001) have
been developed, including Astronomical Dataset Markup Language (ADML, 2001), Advertising
XML (adXML, 2001), Biopolymer Markup Language (BIOML, 2001), Genome Annotation
Markup Elements (GAME, 2001). The increased availability of XML in various domains makes
it a good choice for an integration pivot metamodel for the design and translation of
interoperating system schemas. As a metamodel, XML allows users to define schemas in the
form of XML DTD (Document Type Declaration) or grammars that are uniform syntactic
elements for representing the conceptual characteristics of information systems. Using XML
reduces the complexity of reconciling structural heterogeneity among systems. For example, in
pre-integration data model translation step, the number of required translation tools can be
reduced from $O(N^2)$ to $O(N)$. Several major database system providers have added XML
capabilities to their products. However, XML-enabled DBMS (Data Base Management System)
market is emerging, representing less than 1% of the total DBMS market ($77 millions (IDC,
2001)). The leaders of this emerging market are Software AGS (40.5% market share) followed
by Corp eXcelon (23.3% market share), Compute Associated International Inc. (19.4%), and
Poet Software Corp. (1.3%). Three main approaches have been used to provide XML capabilities
in databases: Non-XML-native information systems use traditional data models to represent
XML documents, XML-native systems are specifically designed for the manipulation of XML
documents and XML-based legacy systems provide XML layers atop traditional information
systems.
XML based data integration plays an important role in B2B application design. This importance stems from several facts. First, as a standard XML facilitates the identification of correspondence and conflicts among different components of cooperating applications. For instance, Louise Lane et al point out that e-commerce companies sell similar products and yet represent these products with different XML schema and ontologies. When several such companies merge and restructure their applications, the integration of the respective information systems and applications into uniform and homogeneous systems requires novel XML related integration tools and methodology. Second, e-commerce applications often involve comparison-shopping in which similar data are searched from different sources and presented to end users, requiring ontologies and semantic based tools to identify similar concepts and elements from different schema or information sources. Finally, the coordination of information systems that are used process client orders and support manufacturing systems, creates the needs for wide acceptance of relatively few XML like standards and related methodology to merge heterogeneous information sources.

**Background and contribution**

In the following section, several traditional interoperability approaches are reviewed, emphasizing the role of semantic resolution and translation tools. Next, interoperability in web oriented environment are discussed, with an emphasis on the important role played by XML in the design of tools for interoperable architectures. The definition of the interoperable architecture is illustrated in B2B architecture development.

In the last ten years or so, several interoperability approaches have been proposed in the literature, ranging from the earlier work on database integration (Batini C., Lenzerni M., 1983) to recent ontology and semantic Web modeling approaches. They can be classified as follows based
on the concept or tool used to represent data and reconcile (both structural and semantic) discrepancies among the participants:

1. The database translation approach is a point-to-point solution that uses direct data mapping to resolve data heterogeneity between pairs of databases (Andersson M., 1994; Blaha M, Premerlani W., Shen H., 1994; Cluet S., Delobel C., Siméon J., Smaga K., 1998; Yan L.L., Ling T.W., 1992). This approach is appropriate when the number of participants in the interoperability environment is small. The number of data translators grows with the square of the number of participant information systems.

2. In the standardization approach, the components of the interoperability environment use the same (or standard) data model to represent data and to communicate. The standard model can be a comprehensive metamodel capable of integrating the requirements of the models of the different components (Atzeni P., Torlone R, 1997; Barsalou T., Gangopadhyay D., 1992; Jeusfeld M. A., Johnen U. A., 1994). Using of a standard metamodel reduces the number of data translators (this number grows linearly with the number of components). However, the construction of a comprehensive metamodel is a difficult task.

3. The federation approach consists of an integrated collection of heterogeneous databases in which federation users access and manipulate data transparently without knowledge of data location (Sheth A. P., Larson J. A., 1990). A federation contains a federated schema that integrates the data exported by the federation participants. There are two types of federations. Tightly coupled federations use a global federated schema constructed by federation administrator to combine the schemas of all participants while loosely coupled federation use non global federated schemas created by users or local database administrator to combine relevant schema.
4. The language based multi-database approach consists of a loosely connected collection of databases in which a common query language is used to access the contents of the participating databases (Keim D.A., Kriegel H.P., Miethsam A., 1994; Lakshmanan L.V.S., Sadri F., Subramian I. N., 1993). In this approach, in contrast to distributed and federated systems, the burden of creating the federated schema is placed on the users who must discover and understand the semantics of other information systems.

5. The mediation approach is based on two main components. The first component is the mediator. It is used to create and support an integrated view of data from multiple sources. The mediator provides data discovery support and various query processing services. The second component is the wrapper. It is used to map the local databases into a common federation data model. The wrapper component provides the basic data access functions (Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J., Widow J., 1995).

6. The ontology based approach uses an ontology to provide an explicit conceptualization of the common domain of a collection of information systems (Benslimane D., Leclercq E., Savonnet M, Terrasse M.N, Yétongnon K., 2000). A conceptualization is an abstract description of concepts of a domain and relationships among concepts. Ontology defines a common vocabulary for users of different systems. The basic idea is to provide a common data semantics that is understood and accepted by all participants in the federation. Defining ontology for a domain is a difficult task that often requires merging overlapping ontologies.

The focus of this paper is on design support for web oriented. New interoperability approaches are required to take into account the characteristics of web based information systems in the design of interoperable systems. The development of data and knowledge
representation formats for exchanging information and services among business applications is one important issue that must be addressed when a common (or pivot) model must be specified for pre-integration resolution of structural or syntactic differences among participating information systems. As we stated above, XML is emerging as a de facto universal standard for data description and exchange in the web. A universal meta-language must provide syntactic elements for describing structured documents. Semantic information can be associated with the description elements to customize XML to model different document types.

XML based integration or interoperability can be achieved in two phases. The first phase is devoted to the creation of XML based information systems that are represented in some form of XML model while the second phase concerns the reconciliation of non-structural differences among the XML based information systems. Three major approaches can be used to achieve XML based information systems. **XML-native** systems are specifically designed for modeling and manipulating XML documents. They provide, in addition to the basic functionalities of traditional databases, specific APIs to allow XML databases and applications to access traditional relational databases using JDBC (Java DataBase Connectivity) or ODBC (Open DataBase Connectivity). XML-native DBMS is an emerging technology whose full capabilities has not yet been fully tested for feasibility and efficiency. In the **non-XML-native** approach, traditional database systems are used to model and manipulate XML document structures. The complexity and accuracy of this modeling depends on the similarity between the data model and the hierarchical structure of XML documents. For example, due to the flat structure of the relational table concept is not suitable for representing the semi-structured format and dynamic properties of XML documents. Despite this inadequacy, many relational systems provide some XML capabilities: IBM DB2 Universal Database v7.1 - XML Extender DBMS provides DTD
functions for storing XML documents; Microsoft SQL server 2000 uses BLOB (binary Object of large size) to store XML documents; other relational DBMS such as Oracle9i - XML Developer Kit and DB2 UDBS provide specific XML data types. Object-oriented capabilities can also be incorporated in DBMS for efficient manipulation of XML documents. The third approach comprises XML based legacy systems that provide translation layers at the top of traditional databases or legacy information systems to wrap and convert them to XML formats. The XML based wrapper supports for sharing legacy system schemas, specifying queries and reformatting of results.

Using XML as a common data representation format addresses syntactic heterogeneity issues of information system integration. XML DTD or schemas, represented by XML grammars, can be created for participant systems to facilitate data exchange among them. However, there is also a need for XML based semantic solutions that incorporate semantics in the terms and concepts used in XML grammar description of information systems. In this paper, we propose a methodology that aims to integrate XML description elements (or concepts) into a semantic generalization tree. The generalization tree represents associations or links between semantically similar concepts. Several issues must be addressed when a generalization tree is constructed over a set of XML based concepts: How to organize and place concepts in the tree? How to apply or use relations among concepts of the generalization tree to map XML document from one XML grammar to another? To address the above issues, we present a methodology called X-TIME that can be used to support the integration of information systems. It is based on an extensible XML oriented metamodel and provides tools for data model translation and the design of wrappers or semantic mediators. X-TIME is a semantics oriented metamodel methodology aimed at achieving interoperable information systems characteristics such as
extensibility and composability. X-TIME is based on a set of metatypes which can be used to specify 1) meta-level semantic descriptors found in the major database models (e.g. flat structure relations, entities, objects, classes, associations etc.) and the emerging web oriented models (e.g. XML models, semi-structured data models). The metatypes are organized in a generalization hierarchy to capture their semantic similarities and correlate constituent interoperable data models. An example of data model translation in B2B application is presented to illustrate the X-TIME methodology.

The remainder of the paper is organized as follows. The background and contribution of the paper have been presented in this section. Section 3 presents an overview of the design of a B2B environment based on the X-TIME methodology. The next four sections are devoted to a presentation of the tools that comprise the X-TIME methodology. The tools corresponding to the different layers of B2B platform development are specified. Section 9 concludes the paper and presents ongoing work.

Overview

To construct B2B platform using standard information systems, specific tools and standard data representation format are used to allow the systems to exchange and share information and services. This overview presents the main X-TIME methodology tools. We show how the tools can be used to create integrated XML metamodel components and standard data definition formats for developing B2B data exchange platform.

Figure 1 depicts an example of B2B platforms development using the X-TIME methodology. It consists of three layers: specification layer, grammar integration layer and translation layer. The specification layer is devoted to the description of local schema concepts using the X-Editor tool. It allows local administrators to graphically describe local data model
XML for B2B applications

concepts and map them into Description Logic and Backus-Naur Form (BNF) descriptions. The Description Logic represents the semantic of the concepts and the connection between them. The BNF defines the structure and constraints of the concepts. The **grammar integration layer** is used to build the metamodel. It takes as input a set of description logic and BNF files of local systems and produces as output a metamodel in which the local concepts are mapped to metatypes and classified according to semantic similarities.

Figure 1

*Overview of the layer to build a B2B platform*

![Diagram]

The metamodel is represented by as an XML-schema file. The result of the metatypes classification is an inheritance lattice (Nicolle C., Jouanot F., Cullot N., 1998) created by a subsumption mechanism in which concept definitions are compared pair wise. The metamodel represents a cooperation model tailored to the specific semantics of the set of interoperating databases. The inheritance subsumption mechanism and the resulting metatypes lattice guarantee interoperability extensibility. To add new databases to the interoperable environment, the corresponding new metatypes are compared to existing metatypes to determine the most semantically similar metatypes. The Strategic Hierarchy Builder (SHB) creates the metamodel of the final B2B environment. The goal of the **translation layer** is to create data model translators.
to support information and service exchanges. This layer comprises two tools. The Transformation Rule Builder tool associates transformation or mapping rules to the metamodel while the Translator Compiler creates one or more translators from the intermediate results of the Transformation Rule Builder tool. The generation of translators using these two tools is carried out in three steps: (1) convert the XML documents representing the local schemas to intermediate meta-schema graphs by substituting occurrences of metatypes for the occurrences of source modeling concepts (XML tags), (2) convert the intermediate meta-schema graphs to equivalent meta-schema by using translation rules to carry out instance mapping between pairs of directly linked metatypes of the generalization hierarchy, and (3) use the **Translator Compiler** to map the meta-schema mapped to the target data model. The **Translator Compiler** groups all translations of metatypes from a source data model to metatypes in the target models (Translation Layer). It can generate complete XSLT (XSL Transformations) files for the automatic conversion of a XML document from a source information system to target information systems (Figure 2).

**Figure 2**

*Overview of the B2B Platform using Translators*

The presented methodology can be applied to Web-services to classify and integrate the XML concepts defined in the Web Services Language Description (WSDL, 2001). WSDL uses XML syntax to describe the methods and parameters of Web-services: protocols, servers, ports,
operations, input and output messages format, and returned exceptions. With WSDL, applications using SOAP (SOAP, 2002) can auto-configure Web-service exchanges, masking most of the low-level technical details. WSDL is the equivalent to the Interface Definition Language (IDL) used in CORBA (Common Object Request Broker Architecture) (CORBA, 2001).

The X-Editor tool

The X-Editor tool maps local data models to description logic and BNF (Backus-Naur form) descriptions in which both the semantic and the structural parts of data model concepts can be defined. Local data model concepts are mapped to metatypes. The metatypes are described from an initial generic description logic concept called "metatype" which provides a generic specification for identifier constraints, links and attributes. Figure 3 shows the basic description of the generic metatype while figure 4 presents the definition of a relational metatype based on the generic description logic metatype.

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 users’ defined constraints that are used to restrict meta data constraints associated with the super metatype of which $M$ is a specialization. $P_M$ 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) that do not allow the encapsulation of data and operations into a type. There are two sorts of metatypes: specific and basic metatypes. Each metatype description is given in both BNF and Description Logic. Some examples illustrate the structural description of the metatypes.
Basic metatypes

Basic metatypes are the core of the metamodel. They are used to define the description concepts of major database models. There are five basic metatypes: a generic metatype "Meta", two object-oriented metatypes "Complex-Object" and "MSimple-Object" and two link or association metatypes "MNary-Link" and "MBinary-Link". The topmost metatype in the generalization hierarchy is the metatype, META. It has an empty structure and is defined by META=([], CMeta, []). CMeta comprises a set of data modeling constraints and operations shared by all metatypes. For example, CMeta defines an ID function for uniquely identifying metatype instances \( i \) such as \( \text{ID}(i) = \{K_j\} \), where \( \{K_j\} \) is a set of labels. The unicity property of the identifier ID is defined by the following rules: For each metatype \( M \) and for each label \( K_j \in \text{ID}(M) \), we define \( M.Kj \) a function such as \( \forall M \in \text{META}, \forall i_z \in M, \exists K_j \in \text{ID}(M) \mid M.Kj(i_z) \neq M.Kj(i_z) \).
The metatype **MComplex-Object** represents object or entity with complex structure such as the class type of the object-oriented model, the entity type of the entity-relationship model and the record type of the Codasyl model. 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 formal definition of \( A_{CO} \) in BNF is given by:

\[
A_{CO} := \text{[attribute_list}_{CO}] \\
\text{attribute_list}_{CO} := \text{attribute}_{CO} | \text{attribute}_{CO}, \text{attribute_list}_{CO} \\
\text{attribute}_{CO} := a:D | a:[]_{CO} | a:\{|T\} | a:\{|T(m,n)\}
\]

where

- \( a \) is an attribute name \((a \in A)\),
- \( T := D | \text{attribute_list}_{CO} \),
- \( D := \text{String} | \ldots | \text{Integer} \),
- \( m := 0 | n \)
- \( n := 1 | 2 | \ldots \)

**Example**: Person = \((A_{Person}, C_{Person}, P_{Person})\) with

\[
A_{Person} := \{PersonN#: \text{String}, LastName: \text{String}, FirstName: \{\text{String}\} (1,3), \]
\[
\text{Children: } \{\{\text{FirstName: } \{\text{String}\} (1,3), \text{Birthday: String}\}\} \\
\text{Address: } [\text{Street: String, Town: String, ZIP: String}] \]
\]

\[
C_{Person} := [ ] \\
P_{Person} := [ ].
\]

The component \( C_{CO} \) and \( P_{CO} \) are not redefined at this level, but are inherited from the super metatype META. The description logic definition of the metatype MComplex-Object is given in the following:

\[
'MComplex-Object' := 'metatype' \\
and exactly(1, r_description) \\
and exactly(1, r_description_label) \\
and exactly(1, r_identifying_description) \\
and no[1, r_link] \\
and atleast(1, r_attribute) \\
and exactly(1, r_attribute_label) \\
and exactly(1, r_attribute_type) \\
and exactly(1, r_attribute_cardinality) \\
and all[r_attribute_type, simple_type_attribute]
\]
The metatype **MSimple-Object** is a specialization of MComplex-Object. It can be used to represent flat structure modeling constructs such as the tuple type of the relational model. It is defined by \( SO = (A_{SO}, C_{SO}, P_{SO}) \) 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 \( A_{SO} \) is defined in BNF by:

\[
A_{SO} := [\text{attribute_list}_{SO}]
\]

\[
\text{attribute_list}_{SO} := \text{attribute}_{SO} | \text{attribute}_{SO}, \text{attribute_list}_{SO}
\]

\[
\text{attribute}_{SO} := a:D
\]

Where \( a \in A \) set of labels

**Example**: Car = \( (A_{Car}, C_{Car}, P_{Car}) \) with

\[
A_{Car} := [\text{CarN#}: \text{String}, \text{Type}: \text{String}, \text{Color}: \text{String}, \text{Power}: \text{String}]
\]

\[
C_{Car} := [],
\]

\[
P_{Car} := [].
\]

The components \( C_{SO} \) and \( P_{SO} \) are inherited from the metatype MComplex-Object. The description logic definition of the metatype MComplex-Object is given in the following:

\[
'MSimple-Object':= 'metatype'
\]

and \( \text{exactly}(1,r\_description) \)

and \( \text{exactly}(1,r\_description\_label) \)

and \( \text{exactly}(1,r\_identifying\_description) \)

and \( \text{no}(1,r\_link) \)

and \( \text{atleast}(1,r\_attribute) \)

and \( \text{exactly}(1,r\_attribute\_label) \)

and \( \text{exactly}(1,r\_attribute\_type) \)

and \( \text{exactly}(1,r\_attribute\_cardinality) \)

The Metatype **MNary-Link** is used to model types that represent connections between real world entities. It is defined by \( NL = (A_{NL}, C_{NL}, P_{NL}) \). \( A_{NL} \) are the link attributes. The component \( C_{NL} \) and \( P_{NL} \) are not redefined at this level, but are inherited from the super metatype META. The BNF definition of \( A_{NL} \) is:

\[
A_{NL} := [\text{NL\_structure}]
\]

\[
\text{NL\_structure} := \text{M}: (m_{M}, n_{M}), \text{N} (m_{N}, n_{N}) | \text{M}: (m_{M}, n_{M}), \text{N}: (m_{N}, n_{N}), \text{attribute\_list}_{CO} | \text{M}: (m_{M}, n_{M}), \text{NL\_structure}
\]
Where M and N are instances of MComplex-Object of MSimple-Object and M,N ∈ A,

\[ m_M, m_N := 0 \mid n_M, n_N := 1 \mid 2 \mid \ldots \]

**Example:** ToBuy = (A\textsubscript{ToBuy}, C\textsubscript{ToBuy}, P\textsubscript{ToBuy}) with

\[ A_{\text{ToBuy}} = \{ \text{Person: (0,n), Car: (1,1), Garage: (1,n)} \}, \]
\[ C_{\text{ToBuy}} = \{ \}, \]
\[ P_{\text{ToBuy}} = \{ \}. \]

The description logic definition of the metatype MNary-Link is:

\[
\text{'MNary-Link'} := 'metatype' \\
\text{and exactly(1,}\_r\_description) \\
\text{and exactly(1,}\_r\_description_label) \\
\text{and exactly(1,}\_r\_identifying_description) \\
\text{and all(}\_r\_identifying_description,'OID') \\
\text{and exactly(1,}\_r\_link) \\
\text{and atmost(1,}\_r\_attribute) \\
\text{and exactly(1,}\_r\_attribute_label) \\
\text{and exactly(1,}\_r\_attribute_type) \\
\text{and exactly(1,}\_r\_attribute_cardinality) \\
\]

The Metatype MBinary-Link categorizes binary connections involving pairs of object types. It is a special case of MNary-Link and it is defined by BL = (A\textsubscript{BL}, C\textsubscript{BL}, P\textsubscript{BL}). The relationships between basic metatypes are shown in Figure 5. The A\textsubscript{BL} component is defined by:

\[ A_{\text{BL}} := [\text{BL}\_structure] \]

\[ \text{BL}\_structure := M: (m_M, n_M), N: (m_N, n_N) | M: (m,n), N: (m,n), \text{attribute_listOC} \]

where M and N are instances of MComplex-Object of MSimple-Object and M,N ∈ A,

\[ m_M, m_N := 0 \mid n_M, n_N := 1 \mid 2 \mid \ldots \]

**Example:** ToWork = (A\textsubscript{ToWorks}, C\textsubscript{ToWork}, P\textsubscript{ToWork}) with

\[ A_{\text{ToWork}} := \{ \text{Person: (0,n), Factory: (1,n)} \}, \]
\[ C_{\text{ToWork}} = \{ \}, \]
\[ P_{\text{ToWork}} = \{ \}. \]

The description logic definition of the metatype MBinary_Link is given by:

\[
\text{'MBinary-Link'} := 'metatype' \\
\text{and exactly(1,}\_r\_description) \\
\text{and exactly(1,}\_r\_description_label) \\
\text{and exactly(1,}\_r\_identifying_description) \\
\text{and all(}\_r\_identifying_description,'OID') \\
\text{and exactly(1,}\_r\_link) \\
\text{and all(}\_r\_link,'binary') \\
\text{and atmost(1,}\_r\_attribute) \\
\text{and exactly(1,}\_r\_attribute_label) \\
\text{and exactly(1,}\_r\_attribute_type) \\
\text{and exactly(1,}\_r\_attribute_cardinality) \\
\]

The specific metatypes capture the semantics of different data models. The initial basic metatype hierarchy can be extended to incorporate the specific metatypes corresponding to major database models. Figure 5 depicts an extension of the basic metatype hierarchy to include the relational and object oriented data models. For example, the table or relation concept of the relational data model is represented by the metatype **MRELATION**. It is defined by

\[ \text{REL} = (A_{\text{REL}}, C_{\text{REL}}, P_{\text{REL}}) \]

MRelation specializes the metatype MSimple-Object. Thus, the structure \( A_{\text{REL}} \) is the same as the structure of \( A_{\text{SO}} \). MRelation refines the ID function defined in META to correspond to the precise definition of the relational primary key; namely, a subsequence of attribute labels that uniquely identify the tuples of a relation. Moreover, since a relation may in some cases represent a relationship between two or more real world entities, MRelation also specializes the metatype MBinary-Link. Therefore, an edge is added between metatypes MRelation and MBinary-Link, and the component \( C_{\text{REL}} \) of MRelation must include constraints inherited from the links metatypes.

The BNF description of the metatype MRelation is:

\[ A_{\text{REL}} := [ \text{attribute_list}_{\text{REL}} ] \]

\[ \text{Attribute_list}_{\text{REL}} := \text{attribute}_{\text{REL}} | \text{attribute}_{\text{REL}}, \text{attribute_list}_{\text{REL}} \]

\[ \text{Attribute}_{\text{REL}} := a: D \text{ where } a \in A \]

The constraints of \( C_{\text{REL}} \) are defined by three axioms.

**Axiom 1**: The non-null property of a relational primary key is defined by:

\[ \forall R_1 \in REL, \exists A_i \in R_1, \forall F_k \in \text{ID}(R_1), R_1.F_k(A_i) \neq \text{NULL} \]

**Axiom 2**: An identifier is made of attributes, which determine in a single way the non-key attributes of the relation:
∀R_{1} \in REL, \forall A_{1}, A_{2} \in R_{1}, \forall F_{k} \in \text{ID}(R_{1}), R_{1}.F_{k}(A_{1}), R_{1}(A_{1}) \rightarrow R_{1}(A_{2})

**Axiom 3:** the notion of external identifier where the key attributes of an instance R1 determine in a single way a sub-sequence of attributes of an instance R2 is defined by:

∀R_{1}, R_{2} \in REL, \forall A_{1}, A_{2} \in R_{1}, \forall A_{2} \in R_{2}, \forall F_{k1} \in \text{ID}(R_{1}) \text{ with } R_{1}.F_{k1}(A_{1}), \forall F_{k2} \in \text{ID}(R_{2})

Where \( R_{2}.F_{k2}(A_{2}), R_{1}(A_{2}) \subset R_{2}(A_{2}) \)

**Figure 4**

*Snapshot of the X-Editor for the MRelation metatype*

The complete definition of the metatype MRelation is presented below. The corresponding snapshot in the X-Editor is given in figure 4.

'\text{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)

'\text{MRelation}_C' := 'metatype'
and exactly(1, r_link)
and all(r_link, 'binary')
and all(r_link, 'foreign_key')
Figure 5 also depicts the inclusion of Object-Oriented model concepts in the initial metatype hierarchy. The concepts of the Object-Oriented model are represented by the metatypes MClass and MInheritance-Link. MClass models the static (attributes and links) and the behavioral (methods) aspect of an object. It is defined by MCLA = (ACLA, CCLA, PCLA). The structure ACLA combines the structures of ACO and ABL. The inheritance from ABL allows the representation of reference attributes. MClass refines the component PCO to introduce the behavioral aspect of the objects. A detailed and formal description of methods is beyond the scope of this paper. Thus, the component PCLA is not presented. CCLA is empty. The component ACLA is defined by:

\[ A_{\text{OO}} := [\text{attribute_list}_{\text{OO}}] \]
\[ \text{attribute_list}_{\text{OO}} := \text{attribute}_{\text{OO}} | \text{attribute}_{\text{OO}}, \text{attribute_list}_{\text{OO}} \]
\[ \text{attribute}_{\text{OO}} := a:D | a:|\text{attribute_list}_{\text{OO}}| | a:\{T\} | a:\{T(m,n)\} \]

Where a is an attribute name (a ∈ A),

\[ T := D | \text{attribute_list}_{\text{OO}} | t' \]
\[ D := \text{String} | \ldots | \text{Integer}, \]
\[ m := 0 | n \]
\[ n := 1 | 2 | \ldots \]

\[ t' \text{ is an instance name of MClass.} \]

The MInheritance-Link is a binary link. In addition to the constraints inherited from metatype MBinary-Link, it maintains a subset constraint between the population of the specialized and generalized metatypes, which generalize the concept of inheritance of the object, oriented data model. Moreover, MInheritance-Link specializes the structure of MBinary-Link to
represent connection without attribute. The metatype MInheritance-Link is defined by \( \text{MIHL} = (A_{\text{IHL}}, C_{\text{IHL}}, [ ]) \). The component \( A_{\text{IHL}} \) is defined by:

\[
A_{\text{IHL}} := [\text{IHL}\_\text{structure}]
\]

\[
\text{IHL}\_\text{structure} := M: (0,1), N: (1,1)
\]

Where \( M \) and \( N \) are instances of MClass

Figure 5

The resulting metamodel

*Strategic Hierarchy Builder (SHB) Tool*

The Strategic Hierarchy Builder is used to construct a generalization hierarchy from the formal description logic specification of the metatypes created by the X-Editor tool. A detailed presentation of description logic, which is derived from the KL-ONE family, can be found in (Woods W.A., Schmolze J.G., 1992).

The resulting description logic specifications are classified into a generalization hierarchy to define an appropriate interoperability metamodel. Automatic classification of metatypes guarantees that the metamodel is updated and reorganized when new data models are added to the interoperability environment. This can be done in three steps. First, the concepts of data models are mapped to description logic structures. Then, a subsumption mechanism is applied to the description logic structures to analyze and place them in the existing generalization.
XML for B2B applications

hierarchy. Finally, the inheritance relationships are used to simplify the initial metatype definitions.

Semi-automatic subsumption and simplification steps are carried out when new information systems and the corresponding data models are added to an interoperable architecture to extend the metamodel by inserting new modeling concepts based on their semantic similarities. This guarantees that a specific metamodel is constructed for the set of interoperating models. From this generalization hierarchy, the Strategic Hierarchy Builder generates the corresponding XML-schema by creating an XML definition for each metatype (XML-schema, 2001). The semantic description of metatypes is used to aid the generation XML-Schema structures for each meta-type. A metamodel for relational and object-oriented models is presented in Figure 4. The corresponding XML-Schema is given in the appendix. A detailed presentation of the metatypes and the subsumption mechanism can be found in (Nicolle C., Cullot N., Yétongnon K., 1999).

Transformation Rule Builder (TRB) Tool

The Transformation Rule Builder is used to create schemas and data translators for interoperable environments. To carry out schema translation, transformation rules are coupled with the generalization links of the metatype hierarchy to convert instances between two connected metatypes. These rules allow instance transformations between two metatypes. They are formally expressed in first order logical predicate and have the general form R(I1, M1, I2, M2), where I1 is the source meta-schema, I2 is the target meta-schema, and M1 and M2 are metatypes. This rule produces the target meta-schema I2 from the source meta-schema I1 by converting all instances of metatype M1 in I1 into one or more instance(s) of metatype M2.
For example, in Figure 6, R(IMSs, CO, IMSt, SO) is a basic object-oriented rule which translates all the MComplex-Object instances (CO) of a source intermediate meta-schema (IMSs) to instances of the metatype MSimple-Object instances (SO) in a target intermediate meta-schema (IMSt).

Figure 6
*Example of transformation rule between a pair of directly linked metatypes*

Basic rules are applied are associated with basic metatypes while extended rules handle one or more non-basic metatypes. The knowledge of both hierarchy and deducted descriptions of metatypes reduce the cost of the rule definition process. In our example, the resulting hierarchy presents a metatype MRelation that inherits from MSimple-Object and MBinary-Link metatypes (Figure 7). Thus, three transformation rules must be defined: One to transform instances of the metatype MRelation into the instances of MSimple-Object and MBinary-Link (R1), another rule to transform instances of MSimple-Object into the instances of MRelation (R2) and a final rule to transform MBinary-Link instances into MRelation instances (R3) (Figure 7).

This number of rules is independent of the cooperation size and the number of heterogeneous data models cooperating. Rules are not limited to converting a structure (or constraints) to other structures (or constraints). In some specific cases, rules can convert a structure into programs or constraints and inversely. Thus, the translation of a relational schema with key constraints in an object-oriented model gives an object schema with encapsulated
methods. The files resulting from the TRB process are XSLT style-sheets (XSLT, 1999) that convert XML documents from a source metatype to a target metatype.

Figure 7

Example of transformation rules between MRelation, MSimple-Object and MBinary-Link

The Translator Compiler

The construction of translators is done in three steps. The first step establishes transformation paths between pairs of distant metatypes in the metamodel.

For example, the translation path between the metatype MRelation and the metatype MCLASS is given by MRELATION $\rightarrow$ MSIMPLE-OBJECT $\rightarrow$ MCOMPLEX-OBJECT $\rightarrow$ MCLASS. The transformation path and meta-schema transformation are stored in a schema translation library. Many studies have shown that using a combination of translation rules to map data models can reduce translation cost or complexity. Moreover, a translation path between two concepts can be reused partially or totally. The construction of transformation paths is achieved using Dijkstra's algorithm (Dahl O. J., Dijkstra E. W., Hoare C. A. R., 1973). The second step consists in regrouping transformation paths to create a translation path. This step is achieved by successive amalgamations of transformation paths. First, transformation paths that possess the same source metatype and the same target metatype are regrouped. Next, paths between metatypes generalizing a source model and metatypes generalizing a target model are regrouped. In our example, the translation path between a relational model (MRelation) and an object model (MClass, MInheritance-Link) is [MRelation $\rightarrow$ [MSimple-Object, MBinary-Link] $\rightarrow$
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 optimization of this code is made. The Translator Compiler provides an XSLT style-sheet that allows the translation of a source XML Document into a target one and the Java source code of the translator that allows the combination of XML document and XSLT style-sheet. Therefore, it is possible to build the corresponding executable file on the various platforms composing the cooperation. Using the metatype hierarchy and transformation rules for database interoperation makes the definition of new transformation rules easier because the definitions of two metatypes directly linked are closed. This method of transformation is independent from the number of heterogeneous models in the federation. Transformations are defined between concepts and not between models.

Figure 8

XML-grammar construction Process
To complete the definition of the resulting translators we associate specific XSLT style-sheet to each cooperating systems. These style-sheets are used to represent the final translated XML document into the target database syntax.

Table 1

*Oracle creation script*

```sql
DROP TABLE PRODUCT CASCADE CONSTRAINTS;
DROP TABLE MACHINE CASCADE CONSTRAINTS;
DROP TABLE MANUFACTURE CASCADE CONSTRAINTS;

CREATE TABLE PRODUCT
( PRODUCT_NUM NUMBER(4) NOT NULL,
  PRODUCT_NAME VARCHAR(32),
  PRODUCT_COST NUMBER(13,2),
  CONSTRAINT PK_PRODUCT PRIMARY KEY (PRODUCT_NUM));

CREATE TABLE MACHINE
( MACHINE_NUM NUMBER(4) NOT NULL,
  MACHINE_NAME VARCHAR(32),
  CONSTRAINT PK_MACHINE PRIMARY KEY (MACHINE_NUM));

CREATE TABLE MANUFACTURE
( MACHINE_NUM NUMBER(4) NOT NULL,
  PRODUCT_NUM NUMBER(4) NOT NULL,
  CONSTRAINT PK_MANUFACTURE PRIMARY KEY (MACHINE_NUM, PRODUCT_NUM));

ALTER TABLE MANUFACTURE ADD
( CONSTRAINT FK_MACHINE FOREIGN KEY (MACHINE_NUM)
  REFERENCES MACHINE (MACHINE_NUM));

ALTER TABLE MANUFACTURE ADD
( CONSTRAINT FK_PRODUCT FOREIGN KEY (PRODUCT_NUM)
  REFERENCES PRODUCT (PRODUCT_NUM));
```

*XML-Grammars Generation*

To complete the description of our solution, we detail in this section the automatic generation of XML-grammars from the specification of legacy database concepts. The Figure 8 represents the different levels of abstraction in a database and the corresponding grammars and documents used to represent them in an XML format. There are four levels of abstraction: metamodel, model, schema and data that are structured on two layers (local and cooperative). The local layer represents the schema and data that are contained in the database.

The cooperative layer represents all levels of abstractions. At this layer, XSLT style-sheets (noted SS in a circle in the Figure 8) are used to pass from an abstraction level to another
one. These XSLT style-sheets transform a source XML document into a target XML document (for the change of abstraction level). Similarly, XSLT style-sheets are used to convert from the cooperative layer to the local layer by taking into account the specific syntax of local data models to format the XML document. This conversion is made using a set of case-tools and a metamodel. Below we present the different levels of the cooperation layer: metamodel, generic model and XML-grammar. To illustrate this process, we use the example given in Table 1. This example is an Oracle creation script. The script talks about machines, which make products.

Table 2

XML-Grammar of a generic relational model

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Relation">
    <xs:sequence>
      <xs:element name="Primary_Key">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Attribute" minOccurs="1" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Attribute_Label" type="Attribute_Label"/>
                  <xs:element name="Simple_Attribute_Type" type="Simple_Attribute_Type"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Foreign_Key" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Attribute" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Attribute_Label" type="Attribute_Label"/>
                  <xs:element name="Foreign_Table" type="ComponentName"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The metamodel level includes a set of tools and a metamodel for the creation of generic concepts (called metatypes). The structure of these metatypes is described in description logic.
and in XML-grammar (appendix). The instantiation of these concepts generates a specific XML-grammar that represents the concepts of a generic data model (i.e. Relational models, Object-Oriented models with no industrial DBMS correspondences). This generic model is used at the model level to represent specific data models that are derived from this model (for example the relational model generalizes those used by Oracle, MySQL, DB2…). An example of this process is presented in Table 2. This table presents an XML-Grammar of a generic relational model.

The XML-grammar defined at this level is derived from the metamodel level and allow the generation of XML documents that describe the syntax of specific models (corresponding to various DBMS). The XML-Grammar presented in Table 2 is translated into a specific XML-grammar for Oracle DBMS. The result of this process is described in Table 2. With XSLT style-sheets, these documents are used to generate the XML-grammar at the schema level. This XML-grammar represents the various database schemas formatted into the XML documents.

Table 3
XML-Grammar of a specific relational schema

```
<schema>
  <element name="ORACLE-SCHEMA">
    <element ref="TABLE" occurs="ONE-ORMORE" />
  </element>
  <element name="TABLE">
    <element ref="TABLE_NAME" occurs="REQUIRED" />
    <element ref="PRIMARY_KEY" occurs="REQUIRED" />
    <element ref="ATTRIBUTE" occurs="ZEROORMORE" />
  </element>
  <element name="PRIMARY_KEY">
    <element ref="ATTRIBUTE" />
  </element>
  <element name="ATTRIBUTE">
    <element ref="ATTRIBUTE_NAME" occurs="REQUIRED" />
    <element ref="ATTRIBUTE_TYPE" occurs="REQUIRED" />
    <element ref="FOREIGN_KEY" occurs="OPTIONAL" />
  </element>
  <element name="FOREIGN_KEY">
    <element ref="TABLE_NAME" occurs="REQUIRED" />
  </element>
  <element name="TABLE_NAME" content="mixed" />
</schema>
```
The specific relation XML-grammar of Table 3 can be used to transform the example schema of Table 1 into a valid XML document. The resulting XML document is presented in Table 4. In a cooperative environment, this document is used to exchange the view of the local databases between the cooperating information systems.

Table 4

XML document of a relational schema

```
- <ORACLE-Schema>
  - <TABLE>
    - <TABLE_NAME>MACHINE</TABLE_NAME>
    - <PRIMARY_KEY>
      - <ATTRIBUTE>
        <ATTRIBUTE_NAME>MACHINE_NUM</ATTRIBUTE_NAME>
        <ATTRIBUTE_TYPE>NUMBER(4)</ATTRIBUTE_TYPE>
      </ATTRIBUTE>
      - <ATTRIBUTE>
        <ATTRIBUTE_NAME>MACHINE_NAME</ATTRIBUTE_NAME>
        <ATTRIBUTE_TYPE>VARCHAR(32)</ATTRIBUTE_TYPE>
      </ATTRIBUTE>
  </TABLE>
  - <TABLE>
    - <TABLE_NAME>MANUFACTURE</TABLE_NAME>
    - <PRIMARY_KEY>
      - <ATTRIBUTE>
        <ATTRIBUTE_NAME>MACHINE_NUM</ATTRIBUTE_NAME>
        <ATTRIBUTE_TYPE>NUMBER(4)</ATTRIBUTE_TYPE>
      </ATTRIBUTE>
      - <FOREIGN_KEY>
        <TABLE_NAME>MACHINE_TYPE</TABLE_NAME>
      </FOREIGN_KEY>
  </TABLE>
  - <TABLE>
    - <TABLE_NAME>PRODUCT</TABLE_NAME>
    - <PRIMARY_KEY>
      - <ATTRIBUTE>
        <ATTRIBUTE_NAME>PRODUCT_NUM</ATTRIBUTE_NAME>
        <ATTRIBUTE_TYPE>NUMBER(4)</ATTRIBUTE_TYPE>
      </ATTRIBUTE>
      - <FOREIGN_KEY>
        <TABLE_NAME>PRODUCT</TABLE_NAME>
      </FOREIGN_KEY>
  </TABLE>
</ORACLE-Schema>
```
From the XML document presented in Table 4, we can produce the XML-grammar to exchange data through the network in XML documents. The Table 5 presents the result of the transformation of the XML document of a relational schema into an XML-grammar of relational data.

Table 5

**XML-Grammar of relational data**

```
<schema>
  <element name="ORACLE-DATA">
    <element ref="MACHINE" occurs="ZEROORMORE" />
    <element ref="PRODUCT" occurs="ZEROORMORE" />
    <element ref="MANUFACTURE" occurs="ZEROORMORE" />
  </element>
  <element ref="MACHINE">
    <element ref="MACHINE_NUM" occurs="REQUIRED" />
    <element ref="MACHINE_NAME" />
  </element>
  <element ref="PRODUCT">
    <element ref="PRODUCT_NUM" occurs="REQUIRED" />
    <element ref="PRODUCT_NAME" />
    <element ref="PRODUCT_COST" />
  </element>
  <element ref="MANUFACTURE">
    <element ref="MACHINE_NUM" occurs="REQUIRED" />
    <element ref="PRODUCT_NUM" occurs="REQUIRED" />
    <element ref="MACHINE_NUM" content="mixed" />
    <element ref="MACHINE_NAME" content="mixed" />
    <element ref="PRODUCT_NUM" content="mixed" />
    <element ref="PRODUCT_NAME" content="mixed" />
    <element ref="PRODUCT_COST" content="mixed" />
  </element>
</schema>
```

Table 6

**Example of SQL query**

```
SELECT *
FROM MACHINE, MANUFACTURE, PRODUCT
WHERE MANUFACTURE.MACHINE_NUM = MACHINE.MACHINE_NUM
AND MANUFACTURE.PRODUCT_NUM = PRODUCT.PRODUCT_NUM
AND PRODUCT.PRODUCT_COST >= 1999.99;
```

The grammar presented in Table 5 is used to generate document that contains data, which result from queries. An example of XML document is given in Table 7. This document is the result of the following query (Table 6) sent to the local database.
Conclusion

In this paper, we have presented an XML based integration approach and toolkit for B2B applications called X-TIME. It uses an adaptable semantics oriented metamodel that defines a set of meta-types for representing meta-level semantic descriptors of data models that can be found in Web sites. The meta-types are organized in a generalization hierarchy to capture semantic similarities between modeling concepts and correlate constituent data models of interoperable systems. The solution allows the creation of cooperative environments for B2B platforms and provides the first layers for the development of a Web services integration platform. We propose a set of tools based on the metatype hierarchy to support the development of B2B applications in cooperative or interoperable architectures. The toolkit provides different tools for automating the construction of the metatype generalization hierarchies, for defining and associating transformation rules to the paths of a metatype generalization hierarchy and for data model translators that are required by an application to map a web format into another representation model. Our future work will focus on using web enable semantic tools for an integration of web services based on the semantic descriptions of the services.
Appendix

W3C XML schema of X-Time DL

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- Simple Type Definition -->
  <xsd:simpleType name="ComponentName">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="Attribute_Label">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="Simple_Attribute_Type">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="Min_Multivalued_Cardinality">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="Max_Multivalued_Cardinality">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="Description_Identifier">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="Description_Label">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <!-- Constructor Definition -->
  <xsd:complexType name="Constructor">
    <xsd:sequence>
      <xsd:element name="ComponentName" type="xsd:string"/>
      <xsd:element name="Complex_Attribute_Type" type="xsd:string"/>
      <xsd:element name="Attribute_Cardinality" type="xsd:string"/>
      <xsd:element name="Attribute_Type" type="xsd:string"/>
      <xsd:element name="Multivalued_Cardinality" type="xsd:string"/>
      <xsd:element name="Attribute_Label" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- Generic Metatype Definition -->
  <xsd:complexType name="Role">
    <xsd:sequence>
      <xsd:element name="MetatypeName" type="xsd:string"/>
      <xsd:element name="Multivalued_Cardinality" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- Links Definition -->
  <xsd:complexType name="Role">
    <xsd:sequence>
      <xsd:element name="Role" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```
```xml
<xs:complexType name="Binary_Link">
  <xs:sequence>
    <xs:element name="Role" type="Role"/>
  </xs:sequence>
  <xs:attribute name="ComponentName" type="ComponentName"/>
</xs:complexType>

<xs:complexType name="N_ary_Link">
  <xs:sequence>
    <xs:element name="Role" type="Role"/>
  </xs:sequence>
  <xs:attribute name="ComponentName" type="ComponentName"/>
</xs:complexType>

<xs:complexType name="Description">
  <xs:sequence>
    <xs:element name="Description_Identifier" type="Description_Identifier" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Description_Label" type="Description_Label" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="ComponentName" type="ComponentName"/>
</xs:complexType>

<xs:complexType name="Metatype">
  <xs:sequence>
    <xs:element name="Description" type="Description" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Link" type="Link" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ComponentName" type="ComponentName"/>
</xs:complexType>

<xs:complexType name="MComplex_Object">
  <xs:sequence>
    <xs:element name="Description" type="Description" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Attribute" type="Attribute" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ComponentName" type="ComponentName"/>
</xs:complexType>

<xs:complexType name="MSimple_Object">
  <xs:sequence>
    <xs:element name="Description" type="Description" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Attribute" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ComponentName" type="ComponentName"/>
</xs:complexType>
```
References


Conference of Entity-Relationship Approach, p 403-419, Manchester UK, December, 1994


http://www.w3.org/TR/soap12-part0/

http://uddi.org/pubs/uddi_v3.htm


http://www.zapthink.com/online/acronyms.htm