Journal of the Association for Information Systems

Information Exchange Between Humanitarian Organizations: Using the XML Schema IDML

Stefan Huesemann
Department of Informatics
University of Fribourg, Switzerland
Stefan.Huesemann@unifr.ch

ABSTRACT

This article explains challenges that arise when humanitarian organizations want to coordinate their development activities by means of distributed information systems. It focuses on information exchange based on the eXtensible Markup Language (XML) and relational databases. This piece discusses how to save hierarchical XML documents in relational databases. It introduces conversion rules to derive a relational database model from XML schemas. The rules are applied for the design of a database for the management of humanitarian development projects. The underlying schema for the database is the International Development Markup Language (IDML). This exchange standard for development-related activities is described. The article gives details on how a traditional relational database can import or export XML documents, i.e. how it can be XML-enabled.

Keywords: International Development Markup Language (IDML), information exchange standard, mapping XML schemas to relational databases, XML-enabled databases, humanitarian development project, AIDA PC

I. INTRODUCTION

Communication or information exchange is the basis of every interaction between different actors. Actors can be people or computer programs. To communicate with each other, actors need a common language and a common understanding of the vocabulary they use.

The focus of this paper is on computer-based information exchange between stakeholders of humanitarian development projects. Humanitarian development projects are activities that have the purpose of helping people to live on a decent level. Humanitarian organizations are institutions that are involved in those activities. Many organizations from various countries are cooperating in such projects. This is a challenge for communication. A great deal of the information exchange is

computer based and runs over the Internet. This raises the issue of what information should be shared and how. There is a tendency to standardize some of the data exchanged to manage development projects. One important emerging standard is the International Development Markup Language (IDML).

Information that is processed by computers should be structured in a way recognizable to the machine. This minimizes the administrative burden of manually reentering data. Exchange standards can be defined with eXtensible Markup Language (XML) schemas [WWW Consortium 2002]. IDML is such a standard.

A problem when working with XML documents is how to store them. One approach is to de-serialize[1] them and to save the data in a database. We will explain how XML schemas can be mapped to relational databases and how data can be exchanged between different databases via the XML. A relational database for the management of development information will be designed to illustrate the theory.

In the next section, we present a classification of information exchange architectures. We then explore how international humanitarian projects work and the problems that occur in the field of information sharing. The IDML will be described as one possible approach to exchanging development related information. Theory on ways of storing XML and the design of relational database models based on XML schemas will be examined. We will apply rules to derive a relational database model from the IDML schema and describe the implementation of the model in a Microsoft Access 2000 database. The same section will also show how the data in the database is exported to an XML file. Two supplementary rules for deriving relational databases from XML schemas are developed and the benefits and limitations of the example will be outlined. Finally, the experience gained in the case study will be examined in a wider context.

II. STATE OF THE ART IN INFORMATION EXCHANGE

We start with an overview of alternatives for data exchange between heterogeneous information management systems (IMS) that have been discussed in the literature. We only consider XML as a data transport format although some of the architectural options have been realized with other formats such as electronic data interchange (UN-EDIFACT).

IMSs can build on a variety of data models (e.g., hierarchical, relational, object-oriented, XML). Raghavan and Garcia-Molina [2001] classified existing architectures for tying different IMSs into three categories:

- layered architectures,
- loosely coupled architectures, and
- extension architectures.

In systems with a layered architecture, an IMS of one type is implemented as an application that operates over an IMS of another type. The main advantage of this approach is that the top-level IMS can leverage the facilities of the underlying IMS without significant additional development time and effort. However, the challenge lies in mapping the data types and operators used by the top-level IMS

[1]Serialization is a process whereby a data term is structured in a simple one-dimensional format. De-serialization is the inverse [Martin et al. 2000, p. 501].

in terms of the types and operators supported by the underlying IMS [Raghavan and Garcia-Molina 2001].

Loosely coupled architectures isolate the integration logic in a separate integration (or mediation) layer. This layer provides a unified access interface to the integrated system using its own data and query languages. The fundamental challenge in this architecture is to design efficient mechanisms to translate queries expressed in the unified model in terms of the query capabilities of the individual IMSs. The advantage is that, unlike the other two architectures, modifications to the individual IMSs are minimal or completely unnecessary. This approach is also known as mediator-wrapper architecture [Papakonstantinou et al. 1998; Thiran and Hainaut 2001].

Finally, extension architectures enhance the capabilities of a particular type of IMS by using an extension module that provides support for new data types, operators, or query languages usually available only in the IMSs of another type. When extension interfaces are available in the original IMS, the extension module can be implemented using these interfaces.

The application case presented later belongs to the latter type of architecture. It includes a module that sits on top of the relational database and is capable of producing XML files that are compliant with the IDML schema.

Any of these architectures can more or less implement functionalities like querying the underlying data, inserting data, or even modifying the data definitions.

A major problem when integrating information from heterogeneous systems is the mapping of the involved data models or schemas [Abiteboul et al. 1999]. Some theory on mapping and a set of rules for converting XML schemas to the relational model will be presented in the section on XML and relational databases.

III. INFORMATION EXCHANGE BETWEEN STAKEHOLDERS OF DEVELOPMENT PROJECTS

Almost anywhere in the world there are people who need substantial help to survive and to live on a minimal standard. That is the reason why humanitarian organizations (such as the Red Cross) exist. Humanitarian organizations are non-governmental organizations (NGOs) that organize programs and projects in two major fields:

- Relief activities (i.e., short term interventions such as refugee support in crises) and
- Development projects (i.e., medium or long term actions with sustainable impact such as education of farmers or the construction of infrastructure).

International funding agencies and private donors give financial support to humanitarian organizations. There is increasing competition for donations between organizations. Although many NGOs have similar objectives, the fight for financial resources discourages them from cooperating. On the other hand, many partners can be involved in a project. In general, one or more funding agencies (e.g., ECHO, USAID, GTZ) and many private donors give money or other resources to international humanitarian organizations. They set up programs which can comprise several projects (e.g., a program for the eradication of malaria). In cooperation with local partners, specific projects are defined and resources are allocated. Local partners can be companies, governments or non-governmental organizations. Sometimes the local partners contract other partners to achieve their goals. The main objective of humanitarian projects is to help needy people.

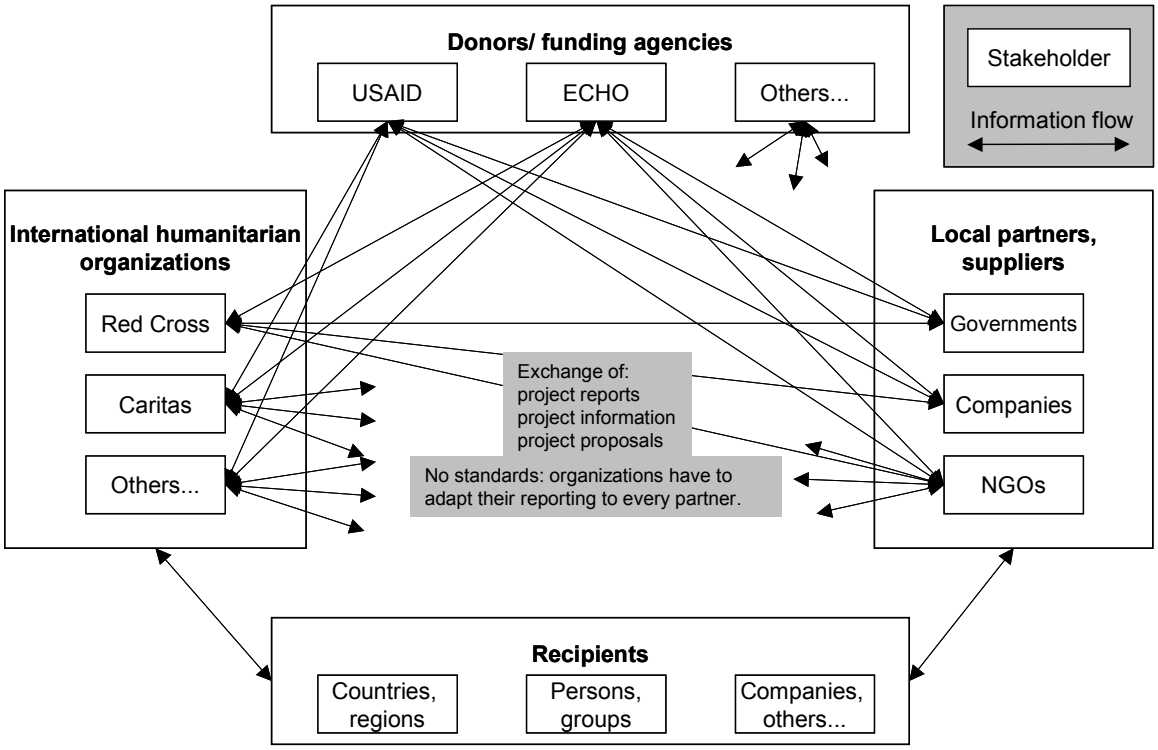

Figure 1. Information Exchange Between Stakeholders of Development Projects

Figure 1 exemplifies the various exchanges between stakeholders of development projects. The rectangles represent types of stakeholders such as donors or international humanitarian organizations. The arrows show information flows. All stakeholders are interested in various facets of information about projects. Many organizations publish facts about their past, ongoing, or planned projects on the Internet. More detailed information such as project reports are exchanged directly between project participants. There is no common standard for the exchange of project information and more detailed reports. (For an analysis of websites that publish information on development projects see Huesemann [2001].)

Although interested people can find certain information on organization websites, it is difficult to get an overview of the activities undertaken by the independent groups in a region. "The effectiveness of foreign aid is impaired by deficient information, fragmentation, and lack of coordination" [Global Development Gateway 2000, p. 6].

THE DEVELOPMENT GATEWAY AND THE AIDA DATABASE

Initiated with the assistance of the World Bank, the Development Gateway is a web portal for development issues. Now a foundation, it is intended as a knowledge-sharing initiative by and for those having a stake in development [Development Gateway 2001, p. 1].

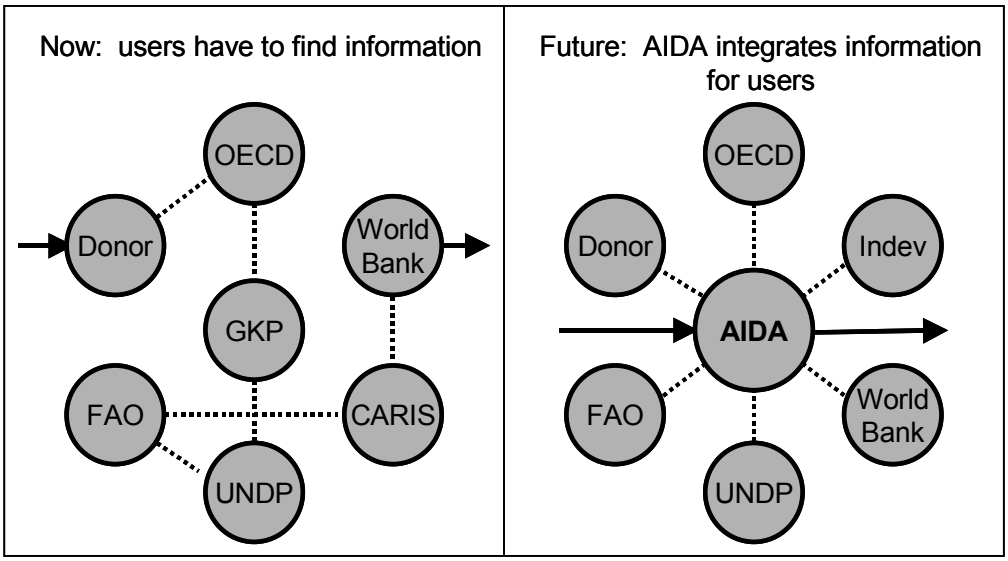

Figure 2. AIDA as Information Integrator [AIDA 2002]

The Accessible Information on Development Activities (AIDA) database [AIDA 2002] of the Development Gateway is an attempt to make information about different organizations and their projects available on one website. The AIDA initiative is a response to the demand for timely and reliable information on who is doing what in various locations, and with what results.

The goal of AIDA is to enable internet users to access information already available on the web sites of development organizations through a common entry point that provides integrated results for major activities by country, sector, source, and operational status, as shown in Figure 2. The circles represent organizations that publish development information on the WWW.

PROBLEMS WITH INFORMATION EXCHANGE

To easily collect data from different websites (harvesting), an exchange standard that fulfills the information needs of the stakeholders of development projects is necessary. The emerging standard for this type of information is the International Development Markup Language (IDML). Various organizations already use IDML for sharing simple project information, among them the Development Gateway.

One problem for stakeholders that want to share information with the AIDA database or any other system that uses IDML as an interface is that they need an application that contains information about their projects. However, experience gained from the AIDA pilot phase demonstrated that even large organizations sometimes lack of such databases.

Furthermore, the technical know-how of development organizations in many cases is not yet sufficient for generating IDML files from their own data.

AIDA PC: A DATABASE FOR SIMPLE PROJECT INFORMATION

Based on these problems, the defined objective was to design a desktop application that would enable a "low-tech" organization to manage basic project information and to exchange this information with others. The main functionality of the resulting application is the ability to export information to an XML file compliant with IDML. This makes sharing information with other systems or organizations easy and the data is reusable by applications that know the exchange standard for development activities.

The database was called AIDA PC because its data model is derived from the IDML version used by the AIDA database of the Development Gateway and because it runs on a personal computer.

The project was carried out in cooperation with the Development Gateway.

HOW DOES THE INFORMATION SHARING PROCESS WORK?

The different architectures for information exchange were outlined earlier when we examined the state of the art in information exchange, but nothing has been said about the exchange process.

Organizations that want to share information electronically with others have different alternatives. They can:

- exchange documents and data directly with partners (bilateral exchange),
- publish their projects as an HTML page on the WWW, or
- make available an XML document on a web server.

The third alternative has the advantage that anybody with access rights can see and retrieve information. It can be easily reused in other applications without human intervention thanks to the mark up.

An organization that wants to share information this way maps its database fields to the IDML elements and publishes the resulting XML file online. A system like AIDA picks up the data from the organization's web server in a regular, automated fashion. This information is then integrated in one view through AIDA.

If an organization's database cannot dynamically generate data in XML, it can post data using the AIDA PC desktop database.

Figure 3 shows the architecture and process of data exchange between two databases via XML. The brackets at the bottom show the tasks implemented with AIDA PC and by the web-based AIDA. AIDA PC can store data in a database and generate IDML-compliant XML files. This is done by the XML generator, which uses mapping and conversion rules applied to the IDML schema. The XML files can then be distributed through various electronic channels. The database that imports the files needs an XML processor that provides access to their content and structure [Martin et al. 2000, p. 424]. An import module could use the document object model (DOM) [WWW Consortium 2002]. DOM loads XML files, validates them against the referenced schema, and builds a tree structure. Error handling is taken care of by the DOM. The elements of the tree can then easily be accessed and imported to the relational database, provided corresponding elements and columns have been mapped.

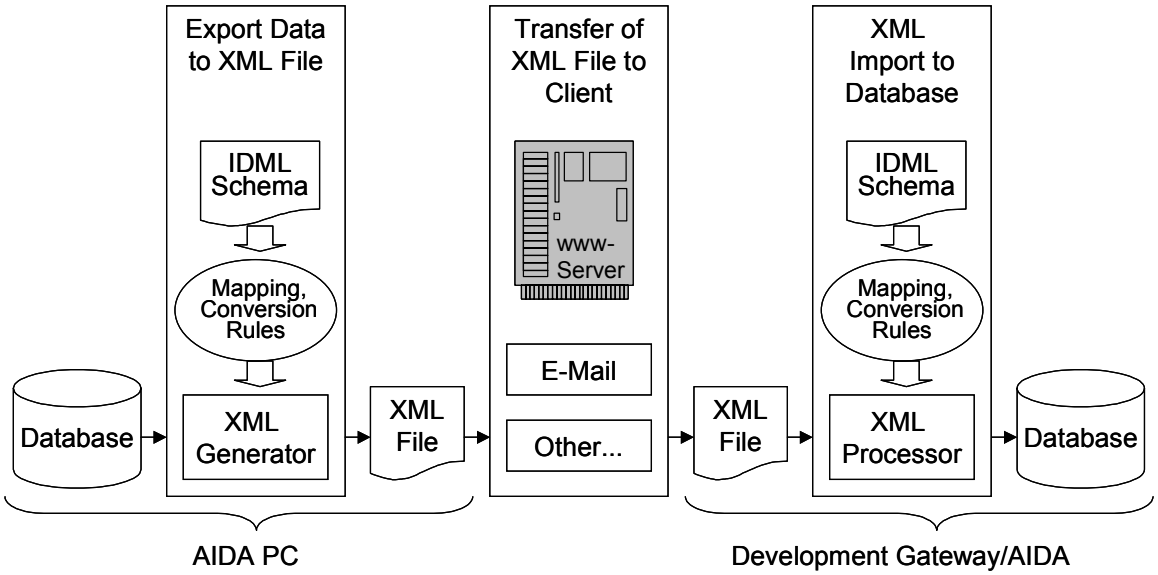

Figure 3. Data Exchange Between Two Databases Using XML

This approach fits the extension architecture category presented earlier. The XML generator is a module implemented with a programming interface offered by database and provides the relational database with certain XML-functionalities.

The following sections are structured in accordance to the rectangle on the left side of Figure 3. First, the IDML schema is described. Rules for mapping XML schemas to relational databases are presented. The XML generator is implemented.

IV. IDML: AN INFORMATION EXCHANGE STANDARD FOR HUMANITARIAN DEVELOPMENT PROJECTS

An information exchange standard is the definition of fields and their meaning. Such a standard is metadata that can serve as an interface between different systems.

More and more standardized information exchange between information systems is done via XML. Using an agreed-upon schema, the systems exchanging data know a “common language.” While the XML schema was recommended as the standard schema definition language for XML [WWW Consortium 2002], other schema languages that fulfill similar purposes exist (e.g., Document Type Definition).

International Development Markup Language (IDML) is the name for an initiative and for the standard this initiative specifies. It is a participatory approach to standards-setting from the international development community. One very active player in this initiative is the non-governmental organization Bellanet, which states the following:

> The IDML initiative proposes to create an open XML standard for international development information—an International Development Markup Language. It also seeks to complement and build on existing development information standards such as the Common Exchange Format for Development Activity Information (CEFDA).... The participating organizations include multilateral and UN organizations, bilateral donors, networks, foundations, NGOs and research institutions. [Bellanet 2000, p. 2]

IDML as an exchange standard is an XML-based set of tags and rules for the types of information unique to the development sector. Elements of the core activity schema include project titles, the organizations involved and their roles, the people involved, funding details, etc. The elements of the schema can be seen in Figure 4. Some of these elements can be found in any type of project. Others, such as like "funding" or "organizations involved," are more specific to international development projects. Funding can come from various donors, and several organizations can be involved. The content of the elements and attributes is probably the biggest difference to profit-oriented projects.

Compared to the situation described in the previous section on information exchange between stakeholders, the expected benefits of IDML include:

- easier cross-platform searching through the WWW,
- enabling different institutions to exchange information in an automated way (IDML as transfer format),
- that organizations can contribute to multi-institution information collections (such as Global Knowledge Partnership, Development Gateway, OECD Creditor Reporting System, etc. [Huesemann 2001]) for better coordination with other groups' activities, and
- connecting internal systems to a public web site and achieve higher visibility of the projects.

The IDML schema is defined with the XML schema syntax. Annotated extracts of IDML follow.[2] The square brackets show where code was left out.

[2]When the AIDA PC database was designed, the IDML working group had not yet defined version 1.0 of the standard. For this reason, we used the latest version of IDML available (version 0.91), which was the one used by AIDA. When referring to IDML here, we are talking about this AIDA IDML. The author was actively involved in the revision and validation of the draft presented by Waser [2000] in XML schema syntax [WWW Consortium 2002].

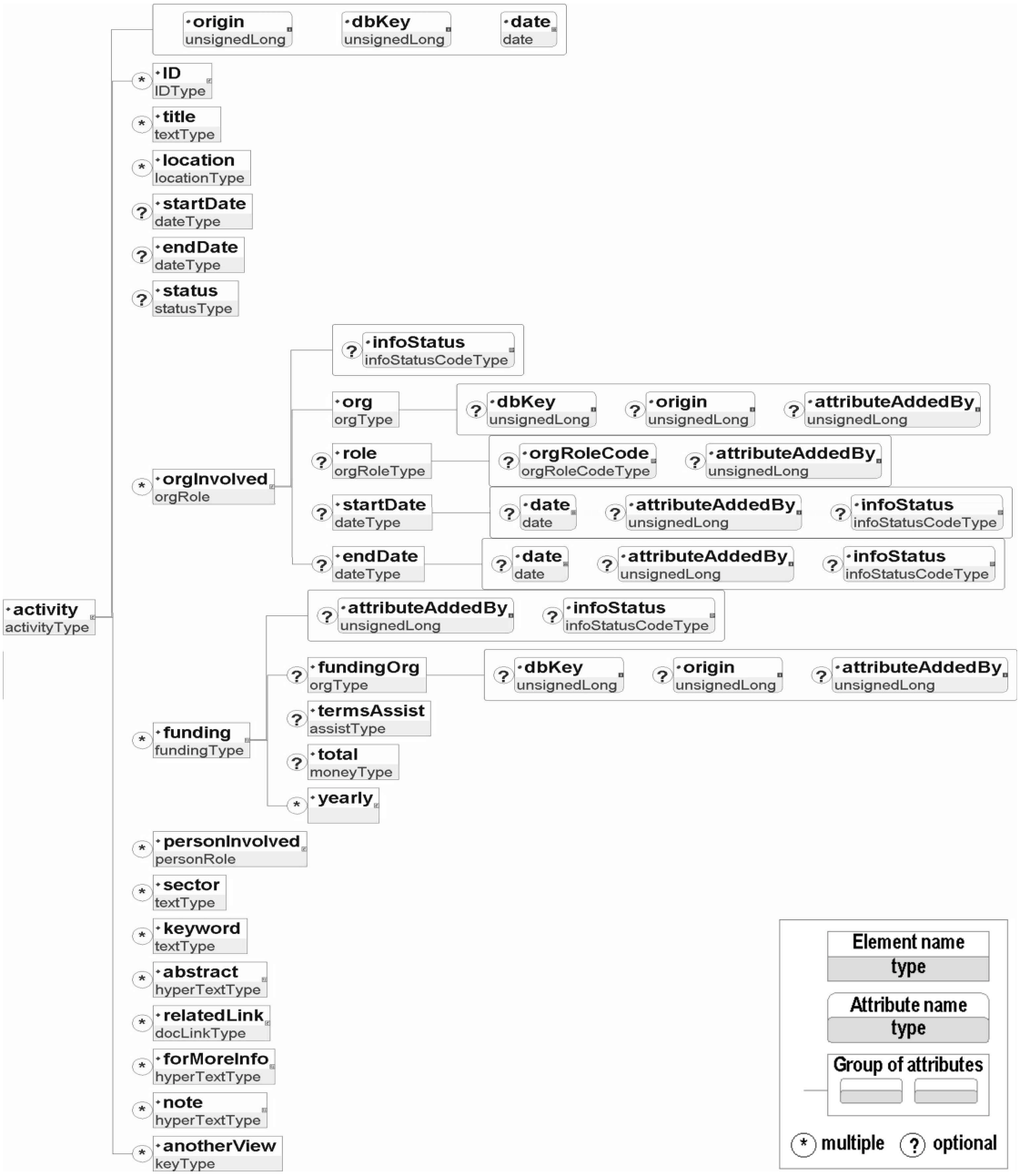

Figure 4. Graphical View of the IDML Schema[3]

[3]The schema editor Turbo XML [Tibco 2001] was used to generate this diagram from the source code.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<schema [...]>
    <include schemaLocation="http://gateway-dev.arsdigita.com/TypeLibs/Currencies"/>
    <include schemaLocation="http://gateway-dev.arsdigita.com/TypeLibs/Countries"/>
    [Import of other sub-schemas]
```

Include imports sub-schemas from other locations. These sub-schemas define types that contain standardized enumerated values (e.g., for currencies).

The **Currencies** file will be described right after the schema. The root element **activities** contains one or more **activity** elements:

```
<element name="activities">
    <complexType>
        <sequence>
            <element name="activity" type="idml:activityType" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
</element>
```

In the next lines, the element **activity** is defined as a complex type **activityType**. An activity can have tree attributes and several elements. Only the **funding** child element is illustrated:

```
<complexType name="activityType">
    <sequence>
        <element name="funding" type="idml:fundingType"
            minOccurs="0"maxOccurs="unbounded"/>
        [other child elements of “activity”]
    </sequence>
    <attribute name="dbKey" type="unsignedLong"/>
    [other attributes]
</complexType>
```

idml:fundingType indicates that the type for the element **funding** is defined in the complex type **fundingType**. The prefix idml specifies that this type is part of the idml namespace. These types can be reused for other elements:

```
    <complexType name="fundingType" mixed="true">
        <sequence>
            <element name="fundingOrg" type="idml:orgType" minOccurs="0"/>
            <element name="termsAssist" type="idml:assistType" minOccurs="0"/>
            <element name="total" type="idml:moneyType" minOccurs="0"/>
            <element name="yearly" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <sequence>
                        <element name="yearStarting" type="date" minOccurs="0"
                            maxOccurs="unbounded"/>
                        <element name="amount" type="idml:moneyType" minOccurs="0"
                            maxOccurs="unbounded"/>
                    </sequence>
                </complexType>
            </element>
        </sequence>
        <attribute name="harmBy" type="unsignedLong"/>
    </complexType>
    [Definition of other complex types]
</schema>
```

Note that the element **yearly** contains detailed information for every year that a certain donor has given funding of a certain type. The schema is finished with the mandatory closing tag.

The following lines show an extract of the sub-schema **Currencies** that defines the simple type **currencyCodeType** with enumerations of possible codes. It is used to limit the possible values of currency attributes.

```
<schema xmlns="http://www.w3.org/2000/10/SMLSchema">
    <simpleType name="currencyCodeType">
        <restriction base="string">
            <enumeration value="USD"/>
            <enumeration value="CHF"/>
            [more enumerations]
        </restriction>
    </simpleType>
</schema>
```

Further information on the IDML initiative and the current state of the consultations can be found at **http://www.idmlinitiative.org/** [IDML 2002].

Figure 4 shows a graphical view of the IDML schema. Elements are represented by rectangular boxes while attributes have rounded corners. Attributes that belong to one element are placed in a rectangle that is linked to the element. Child elements are placed to the right of the parent. Optional elements (occur 0 or one time) are marked with a circled question mark and elements that can occur

any number of times are marked with a circled star. The data type is indicated in the lower part of each box.

In this diagram we only expanded fully the elements **orgInvolved** and **funding**. Other elements also have child elements and attributes. The full schema diagram can be found in Figure A1 in the Appendix.

The root element, not displayed in Figure 4, is called **activities**. It contains at least one **activity**. An **activity** can be any type of development activity i.e., a program, a project, a task, etc.). Activities can be related to each other through the element **anotherView**. The idea is that any contributor of information gives his opinion or view of a project, which might be different from the view of another stakeholder. The same project can hence be described by various **activity** elements. It is a task of the information system that uses this information to display or hide views.[4]

V. XML AND RELATIONAL DATABASES

Now that a possible common language for the information exchange between development organizations has been described,we will discuss how this language can be "taught" to a database. This implies storing and retrieving XML from databases.

STORING AND RETRIEVING DATA

XML documents can be stored as files or they can be decomposed and stored in a database. When talking about XML and databases, two categories are differentiated [DeJesus 2000]:

- XML-enabled databases, and
- native XML databases.

According to the XML:DB Initiative [2002], a native XML database defines a (logical) model for an XML document, stores and retrieves documents according to that model, and has an XML document as its fundamental unit of (logical) storage.

An XML-enabled database is usually a relational or object-oriented database management system. It has an added XML mapping layer provided either by the database vendor or a third party. This mapping layer manages the storage and retrieval of XML data.

In this article, we will only consider XML-enabled relational database management systems (DBMS). Data can originate both from a database (in which case we want to exhibit it as XML) and outside the database (in which case we want to store it in a database) [Bourret 2002]. An example of the former is accounting data about development projects stored in relational databases; an example of the latter are reporting documents written by an executing organization and exchanged with other organizations for coordination.

Three very different approaches exist for saving XML documents in relational tables. The first and simplest is to store an entire XML document as a single database attribute [Raghavan and Garcia-

[4]This very open way of reporting about a project was discussed at the AIDA core working group meeting in Washington,DC, held in May 2001.

Molina 2001]. The second possibility is to interpret XML documents as graph structures and supply a relational schema that can store such graphs [Florescu and Kossmann 1999].

A third approach is to decompose the tagged elements and save them in corresponding fields in the database. This alternative is most useful when working with relational DBMSs because their advantages, such as referential integrity and elimination of redundancies, can be used. The difficulty, however, is to map the XML schema to the tables and columns in the database [Martin et al. 2000, pp. 25, 421].

The third approach will be used for the storage of IDML documents in the AIDA PC database. Importing XML data into a database as well as exporting data from the database to an XML document requires mapping rules. Mapping means that a person or software has to specify what field in a database corresponds to which element or attribute in the XML schema. For example, we might map the <funding> element in IDML to the **funding** table and the <total> child element to the **funding.amount_text** column.

If designing a new database that stores data from XML documents, we might want to derive the model from the XML schema to achieve a maximum of compatibility. This solution was chosen to build the AIDA PC database. Some rules for doing this conversion are described next.

DERIVING RELATIONAL DATABASE MODELS FROM XML SCHEMAS

Rules for deriving one schema from another are here called conversion rules. Mapping rules are used to define corresponding elements in two or more different schemas. There is no one set of standard rules that has been generally accepted by theory and applied in practice for converting and mapping XML schemas to the relational model. XML schemas are quite new, but mapping concepts are known.

Mapping rules and procedures for data exchange can be found in the literature on database migration—e.g., between hierarchical and relational database systems [Gillenson 1990; Meier et al. 1994]—or on information exchange between heterogeneous IMSs [Thiran and Hainaut 2001]. More specific mapping rules between the Unified Modeling Language (UML) and XML schemas can be found in Carlson [2001, p. 194]. Abiteboul et al. [1999] developed methods and tools that support the mapping and translating of data from one representation to another, which includes XML. This theory is useful because the core problem is mapping different data models. However, we need a specific set of rules for the conversion from XML to the relational model.

One way to generate relational models from XML schemas and vice versa is to hardcode a path through the object-relational mapping [Booch et al. 1999; Bourret 2001, 2002]. In this approach, XML is first mapped to an object model and from there to the relational model.

The five mapping and conversion rules presented below are taken from Bourret [2001]. They where chosen for two reasons:

- first, because they allow a direct conversion from an XML schema or document type definition to tables and columns of a relational database, and
- second, because the rules are stated formally.

Similar but less structured conversion rules from XML schemas to relational databases can also be found in Martin et al. [2000].

Rule 1: For each complex element type, create a table and a primary key column.

Rule 2: For each element type with mixed content, create a separate table in which to store the PCDATA, linked to the parent table through the parent table's primary key.

Rule 3: For each single-valued attribute of that element type, and for each singly-occurring simple child element, create a column in that table. If the child element type or attribute is optional, make the column nullable.

Rule 4: For each multi-valued attribute and for each multiply-occurring simple child element, create a separate table to store values, linked to the parent table through the parent table's primary key.

Rule 5: For each complex child element, link the parent element type's table to the child element type's table with the parent table's primary key.

These rules and some of the other sets mentioned earlier have been used to develop mapping algorithms [Collins et al. 2002]. They can be used to convert one schema into another one by a program, but the results usually need human revision, especially when complex structures are involved.

The usual way to design a database would be to make a semantic model first, for example, with an entity-relationship model [Chen 1976]. With the rules presented here, the logic database model (i.e., the tables) is generated directly. However, the application case will show that sticking exactly to these conversion rules does not result in an optimal database design. Two supplementary rules will be formalized.

VI. DESIGN AND IMPLEMENTATION OF THE XML-ENABLED DATABASE AIDA PC

The AIDA PC application case described in this section uses the above rules and shows reasons for not applying them in certain circumstances. The implementation of a simple database extension for the XML generator is presented.

Before starting to design a database, the developer has to know the target user groups and their needs. The AIDA PC database is intended to be used by organizations who do not have a database to manage their project information. They sometimes lack technical infrastructure and know-how; this is why they are referred to here as low-tech organizations.

The database is designed to manage simple project information on development activities. The exchange of project information within or between organizations should be facilitated by the database. The main purpose of the database is to help organizations to upload data to AIDA without using a WWW interface and without knowledge of XML. More information about AIDA PC can be found in Huesemann [2001].

The application is not a project management software in the sense that it has no tools that support the planning and implementation processes of a project.

REQUIREMENTS AND CHOICE OF A DATABASE MANAGEMENT SYSTEM

The choice of a database management system (DBMS) depends on the requirements it should fulfill. Only a few very basic requirements have been defined in that respect. The DBMS should

- be a relational database,
- run on a simple Microsoft (MS) Windows operated PC,
- be in wide-spread use and affordable,
- be XML-enabled and have the possibility of programming an XML export procedure.

MS Access 2000 was chosen as the DBMS to implement the AIDA PC version. Access fulfills the requirements to a fair degree. Unfortunately MS Access 2000 is not XML-enabled.[5] However, it is possible to write Visual Basic modules that handle the import or export of XML documents.

DESIGN OF A RELATIONAL DATABASE DERIVED FROM IDML

The rules suggested by Bourret [2001] for deriving a relational database model from an XML schema will now be applied. The AIDA PC database is modeled from IDML.

We do not have the space to list every element and how it was mapped to the database. Table 1 presents the five rules with an example. The examples relate primarily to the funding element. Later we will explain why we did not follow the rules in certain cases.

The database resulting from the conversion is displayed in the Appendix (Figure A2). It shows the relationship diagram in MS Access (which is not an entity relationship model). For easier readability, Figure 5 shows only the **activity** table, tables related to the funding of a development project, and tables related to **party** and **location**.

At the left side of the diagram in Figure 5, we see the activity table. All child elements of the IDML activity element have been converted to tables (rule 1) and are linked directly to the activity table (rule 5). The tables' columns stand for attributes and elements (rule 3). The name of lookup-tables that hold standardized values for attributes end with **Lookup**. These tables are derived from sub-schemas that are imported to the IDML schema. For instance, the **Currencies** file is converted to the **currencyLookup** table in AIDA PC.

All tables are normalized (third normal form) to prevent redundancy (data duplication). All references between the tables enforce the referential integrity, i.e., the DBMS makes sure referenced values exist and cannot be deleted.

[5]The newer version, Access XP, has XML functionalities. It applies rules to output XML or to import it (table based mapping [Bourret 2001]), but it is not yet possible to map a given schema to the database. As a choice for the implementation of the AIDA PC database, it was no better than the earlier version.

Table 1. Conversion Rules with Examples

Rule	XML Schema	Relational Database
Rule 1: complex element type → table and primary key column.	Element: *activity*	Table name: *activity* Primary key: *dbKey*
	Element: *funding* (child of activity)	Table name: *funding* Primary key: [*activity_dbKey* & *org_dbKey* & *assistType_code*] (composed key)
Rule 2: element type with mixed content → table linked to the parent table through the parent table's primary key.	All mixed contend elements are defined as complex types in IDML version 0.91. ⇒ rule 1 and rule 5	
Rule 3: single-valued attribute of that element type/ singly-occurring simple child element → column in that table.	Element: *total* (child of element *funding*) Attribute: *currency* (of element *total*)	Table name: *funding* Column name: *amount_text* Column name: *currency_code*
Rule 4: multi-valued attribute/ multiply-occurring simple child element → table to store values, linked to the parent table through the parent table's primary key.	Element: *yearly* (multiply-occurring child element of *funding*)	Table name: *funding_yearly* Primary key: [*activity_dbKey* & *org_dbKey* & *assistType_code* & *yearStarting*] Foreign keys to parent table: [*activity_dbKey*, *org_dbKey*, *assistType_code*] (composed key of parent table)
Rule 5: complex child element → link parent and child types' tables with the primary key of the parent table.	Element: *funding* (All child elements of the element activity are complex type elements with mixed content.)	Table name: *funding* Foreign key to parent table: *activity_dbKey*

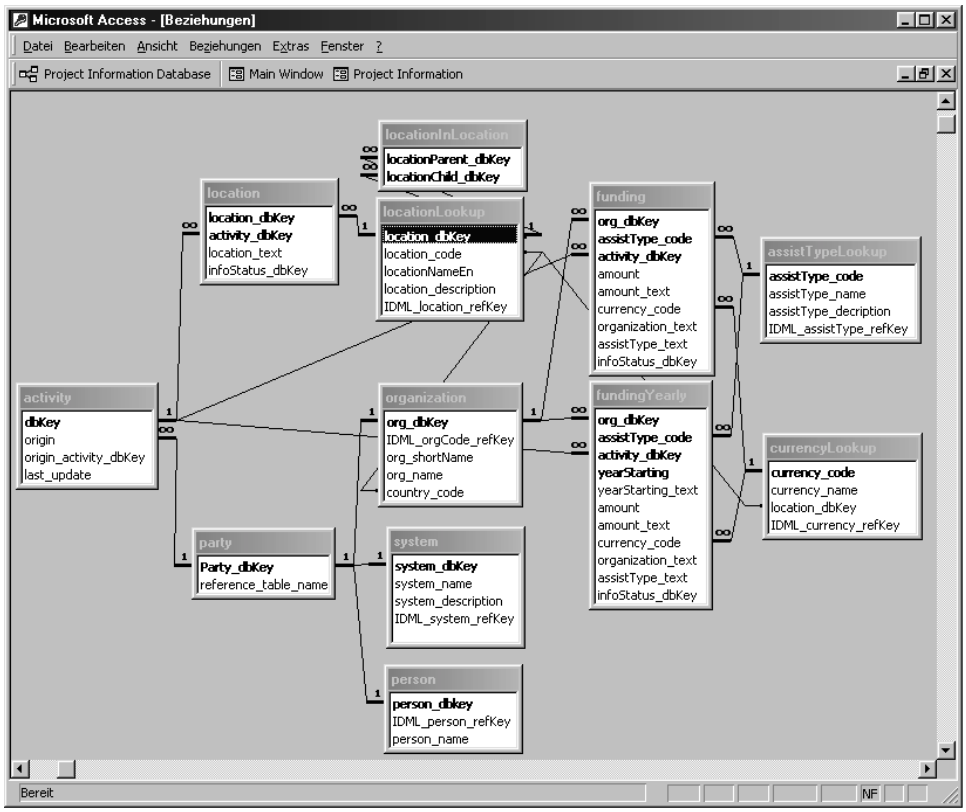

Figure 5. Extract of AIDA PC Relationships Diagram

EXCEPTIONS TO THE CONVERSION RULES

It would have been possible to apply fully the conversion rules from the earlier section, “Deriving Relational Database Models from XML Schemas.” However, the AIDA PC database should provide certain functionalities that cannot be achieved through the strict application of the rules. Another reason is the optimization of the relational database model.

Certain tables in the database are not in the schema. For example:

- The table **party** is a generalization [Smith and Smith 1977] that groups **system**, **organization**, and **person**; it assures the uniqueness of primary keys across these tables. This is necessary because the origin attribute of the activity element can either be a computer system, an organization, or a person.

  Figure 6 is the representation of a generalization [Alhir 1998, p. 77]. Generalizations and specializations are not very well supported in traditional relational databases and the consistency has to be assured by triggers.

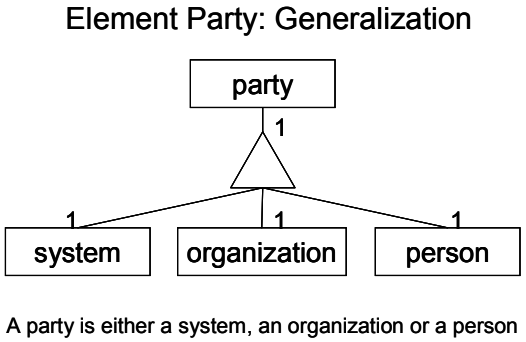

Figure 6. **Party** as Generalization of **System**, **Organization**, and **Person**

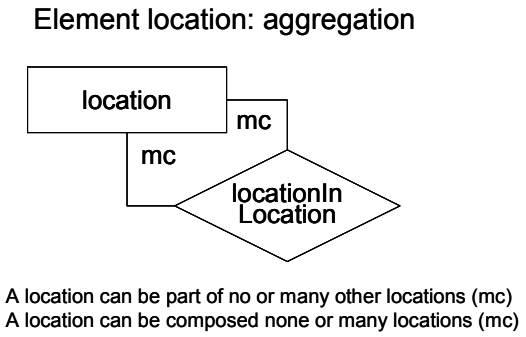

Figure 7. Relationship **locationInLocation** as Aggregation

- The table **locationInLocation** is an aggregation [Smith and Smith 1977]. It gives extra functionality in the database to define dependencies between regions, countries, cities, etc., which are all locations.

  As shown in Figure 7 a location can be part of none, one, or many different other locations (mc). It can also comprise other locations.

- The table **_origin** contains the name of the database owner. This name is referenced when a new development activity is entered in the database. It is used to identify the origin of the project data and to determine reading and writing rights (a user may only modify their own records).

These were examples of tables in the database that were not derived from the IDML schema. On the other hand, certain IDML elements are not found in the database.

- The element **activities** can be omitted because it is a wrapping element and does not have any further attributes [Microsoft 2000]. It is only needed in the schema to allow several **activity** elements in one XML document.

- The **startDate** and **endDate** are grouped together in the table **duration**. This is possible because the two elements have the same data type and their joins to the activity table have the same cardinality. This is done to simplify the relational model.

These examples show that it is not always easy to apply the conversion rules or to program algorithms that automate this process. The more hierarchies, complex types, and sub-schemas an XML schema uses, the more difficult it is to convert it into a relational database model. Depending of the needed functionalities, it also makes sense to simplify (e.g., **duration**) or extend (e.g., **party**) the database model. This does not necessarily make it incompatible with the schema.

Supplementary rules are derived from these observations and will be presented later.

XML GENERATOR

As shown in Figure 3, we need an XML generator to export data from the database. The main functionality of the AIDA PC database is its capability to export information to an IDML file. MS Access 2000 does not provide integrated XML functionalities. There is neither the possibility to map XML schemas to a relational database directly within the schema (an advantage of the XDR schema used by MS SQL-Server 2000 [Microsoft 2000]). This is why the export procedure was programmed as an MS Access Module in Visual Basic 6.

The procedure runs through the data sets in the database with structured query language (SQL) statements and saves the results in a text file. This file can then be exchanged as described earlier (see Section III, subsection “How Does the Information Sharing Process Work?”).

The code displayed below shows extracts of how the **funding** element is generated. The number at the beginning is the line number of the original code. Omitted code is replaced by […]. For more information about the Visual Basic syntax see Sybex [1999] and Britt and Duynstee [2000].

```
6   Public Sub Export_IDML(activity_dbKeys As DOA.Recordset)
    […]
```

The recordset **activity_dbKeys** contains the activities that are to be exported with all the columns in the table **activity** (line 6).

```
25  dbKey = activity_dbKeys(“dbKey”)
    […]
```

The column **dbKey** is saved in the recordset with the same name (line 25). It is used in the SQL queries to project the tuples related to an activity. The following code is applied to every activity in the **dbKey** recordset.

```
114   Set content = db.OpenRecordset(“select * from XML_funding where activity_dbKey =” &
          dbKEY & “ order by org_dbKey, assistType_code”, dbOpenDynaset, dbSeeChanges)
```

For a given activity the corresponding data from the View **XML_funding** is put into the recordset **content**. The recordset holds all columns from table **funding** and related lookup tables like **currencyLookup** (line 114).

XML is a string variable. It holds the output of the queries. Tags are put around the data to transform it into a valid IDML document.

```
115     Dim XML As String
116     Dim fundingYearly As DAO.Recordset
117     Dim fundingOrg, assistType_code As Long
118
119     While Not content.EOF
            […]
123         XML = XML & " <funding attributeAddedBy=""" & content("attributeAddedBy") & """
                infoStatus=""" & content("IDML_infoStatus_refKey") & """>" & vbCr
            […]
132         XML = XML & "       <total amount=""" & content("amount") & """ currency=""" &
                content("currency_code") & """>" & vbCr
133         XML = XML & "       " & content("amount_text") & vbCr
134         XML = XML & "       </total>" & vbCr
135
136         Set fundingYearly = db.OpenRecordset("select * from XML_fundingYearly where
                activity_dbKey =" & dbKey & " and org_dbKey =" & fundingOrg & " and
                assistType_code =" & assistType_code & " order by yearStarting",
                dbOpenDynaset, dbSeeChanges)
137         While Not fundingYearly.EOF
```

For every **funding** entry the corresponding tuples in the **XML_fundingYearly** view are retrieved and saved in the recordset **fundingYearly** (line 136).

Every line in the recordset **fundingYearly** is treated and the **XML** string is further constructed, in a similar way as the **funding** parent element (lines 137-149).

```
138             XML = XML & "       <yearly>" & vbCr
                […]
147             XML = XML & " </yearly>" & vbCr
148             fundingYearly.MoveNext
149         Wend
150         XML = XML & "    </funding>" & vbCr
151         content.MoveNext
152     Wend
        […]
293     End Sub
```

The next entry in the **content** recordset is treated until there is no more funding information for a given activity (lines 119-151). At the end the string, **XML** is written into a text file (XML document).

A very similar procedure could also be executed outside the database by a web-server, for example, in an Active Server Page (ASP). The SQL statements could be sent via open database connectivity (ODBC) to the database. This alternative would generate XML files on demand. In this case, we could classify the architecture as loosely coupled (see Section II), as opposed to the current extension architecture.

The AIDA PC database is shareware and open source in the sense that anybody who can run MS Access 2000 can see or modify the source code. (The source code can be downloaded from **http://www2-iiuf.unifr.ch/is/index.htm?stefanh/diss/** [Huesemann 2001].) The database is currently being tested by the Development Gateway and the International Monetary Fund.

VII. CONTRIBUTIONS TO THEORY AND PRACTICE

The experience gained with the AIDA PC case contributes to the theory and practice of information exchange.

On a theoretical basis, the example shows that the existing conversion rules to derive a relational database model from an XML schema can be optimized for certain cases. They can be extended by at least these two rules:

- XML wrapping elements without any attributes and content (element only) can be omitted in a relational database.
- Mixed-content elements with the same data type and with the same parent element can be grouped in the same table.

These two rules can be added to the five described earlier (see Section V, subsection “Deriving Relational Database Models from XML Schemas”).

Some of the general problems related to mapping existing databases to XML schemas are:

- Schema elements that are missing in a database cannot be mapped. This can be a problem if the fields are mandatory in the schema.
- Information or fields that exist in a database but not in the schema cannot be exchanged correctly.
- Fields in a database can overlap with fields in the schema (i.e., the fields do not correspond exactly). In this case, human intervention is the only good solution.

In practice, the development community can use the study to implement their own information sharing solution or simply use the AIDA PC application. This may minimize software development efforts and even enable low-tech organizations to be more visible on the WWW. The existence and structure of IDML is not well known to the development community. The explanations given above can provide quick insight.

A limitation of the AIDA PC database is its lack of support for the automated import of IDML files. A few challenges with imports are related to synchronization of imported data with already existing entries in the database. Another shortcoming is that the program code has to be changed manually if either side, the database or the IDML schema, changes.

The next steps to be undertaken include the design of an extension to the IDML schema for reporting purposes, providing a set of XSL style sheets to transform and format XML project data to HTML or portable document format (PDF), and the programming of a module that imports IDML files to AIDA PC by using the DOM.

VIII. CONCLUSIONS

The theory presented and the case in which it was applied show how a relational database can be designed to save information from hierarchical XML documents. It illustrates how a traditional database can be XML-enabled and be used to produce XML files that are compliant with an XML schema—in our case, IDML.

Many development organizations operate in regions where even basic infrastructure is missing. In such cases, Prussian officer Carl von Clausewitz's statement, made in his 19th century book about war, applies: even the easiest things are too complicated.

Of course, minimal infrastructure such as electricity and computers are required to use administrative tools like AIDA PC. Critical success factors for AIDA PC and other similar software products for development organizations are ease of use and minimal administrative costs. AIDA PC fulfills these administrative success factors. It was important for the design of the application that the user not need to know anything about databases and XML. This allows the information sharing process to be kept as simple as possible.

The application that was developed is meant to improve the reporting procedures for development activities and make project information available to interested people. In this way, organizational learning and coordination with efforts of other organizations will be improved, which will result in a more efficient use of funds. In the end, however, a bad project will not become a good one because of the software that is used to manage it. Development projects have to be sustainable and fulfill the needs of beneficiaries.

IX. ACKNOWLEDGEMENTS

The work presented here is the result of a close cooperation with the World Bank and the Development Gateway. I am specially grateful to Mark Waser, who actively participated in the design of the AIDA PC database and gave precious advice. I would also like to thank the three reviewers and the editor of JAIS, who made suggestions on how to improve the structure and content of this article.

Editor's Note: This article was received on September 10, 2001. The article was with the author for one month for one revision.

REFERENCES[6]

Abiteboul, S., S. Cluet, T.Milo, P. Mogilevski, J. Siméon, and S. Zohal. "Tools for Data Translation and Integration," *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society* (22:1), 1999, pp. 3-9.

AIDA. "Accessible Information on Development Activities," **http://www.developmentgateway.org/node/100647/**, current as of February 25, 2002.

Alhir, S. S. *UML in a Nutshell*, Sebastopol, CA: O'Reilly & Associates Inc., 1998.

Bellanet International Secretariat. *Exploring the Potential Application of IDML for the Aid Effectiveness Exchange Module of the Global Development Gateway*, Ottawa, ON: Bellanet, 2000.

Booch, G., M. Christerson, M. Fuchs, and J. Koistinen. *UML for XML Schema Mapping—Specification*, Cupertino, CA: Rationale Software Corp. and CommerceOne Inc., 1999.

Bourret, R. "Mapping DTDs to Databases," **http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html**, as of May 2001.

Bourret, R. "XML and Databases," **http://www.rpbourret.com/xml/XMLAndDatabases.htm**, February 2002.

Britt, J.,and T. Duynstee. *Professional Visual Basic 6 for XML*, Birmingham, UK: Wrox Press, 2000.

Carlson, D. *Modeling XML Applications with UML: Practical E-Business Applications,* Boston: Addison-Wesley, 2001.

Chen, P. P. "The Entity-Relationship Model : Toward a Unified View of Data," *ACM Transactions on Database Systems* (1:1), 1976, pp. 9-36.

Collins, S. R., S. Navathe, and L. Mark "XML Schema Mappings for Heterogeneous Data Access," *Information and Software Technology* (44:4), March 2002.

DeJesus, E. X. "XML Enters The DBMS Arena," *Computerworld Online*, **http://www.computerworld.com/cwi/story/0,1199,NAV47_STO53026,00.html**, October 30, 2000.

Development Gateway. *Where Worlds of Knowledge Meet*, Washington DC: World Bank, 2001.

Gillenson, M. L. "Physical Design Equivalencies in Database Conversion," *Communications of the ACM*, August 1990, pp. 9-27.

Global Development Gateway. *Harnessing Knowledge and Technology for Sustainable Development and Poverty Reduction,* Washington DC: World Bank, 2001.

Florescu, D., and D. Kossmann. "Storing and Querying XML Data Using a RDBMS," *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society* (22:3), 1999, pp. 27-24.

---

[6]Editor's Note: The following reference list contains the address of World Wide Web pages. Readers who have the ability to access the Web directly from their computer or are reading the paper on the Web can gain direct access to these references. Readers are warned, however, that

1. These links existed as of the date of publication but are not guaranteed to be working thereafter.
2. The contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions references.
3. The authors of the Web pages, not JAIS, are responsible for the accuracy of their content.
4. The author of this article, not JAIS, is responsible for the accuracy of the URL and version information.

Huesemann, S. "Homepage of Ph.D. Project: Development Information Exchange System," **http://www2-iiuf.unifr.ch/is/index.htm?stefanh/diss/** current as of September 6, 2001.

IDML. "International Development Markup Language," **http://www.idmlinitiative.org/**, current as of February 25, 2002.

Martin, D., M. Birbeck, M. Kay, S. Livingstone, S. F. Mohr, J. Pinnock, B. Loesgen, S. Livingston, N. Ozu, M. Seabourne, and D. Balies. *Professional XML*, Birmingham, UK: Wrox Press, Ltd., 2000.

Meier, A., R. Dippold, M. Mercerat, A. Muriset, J.-C. Utersinger, R. Eckerlin, and F. Ferrara. "Hierarchical to Relational Database Migration," *IEEE Software* (2:3), 1994, pp. 21-27.

Microsoft. *SQL-Server 2000 Help: Writing XML with OPENXML*, Redmond, WA: Microsoft Press, 2000.

Papakonstantinou, Y., A. Gupta, and L. Haas. "Capabilities-based Query Rewriting in Mediator Systems," *Distributed and Parallel Databases* (6), 1998, pp. 73-110.

Raghavan, S., and H. Garcia-Molina. "Integrating Diverse Information Management Systems: A Brief Survey," Working Paper, Computer Science Department, Stanford University, 2001.

Smith, J. M., and D. Smith. "Generalization and Aggregation," *ACM Transactions on Database Systems* (2:2), 1977, pp 105-133.

Sybex Inc.. *Visual Basic 6 Complete*, London: Sybex International, 1999.

Thiran, P., and J.-L. Hainaut. "Wrapper Development for Legacy Data Reuse," *Proceedings of the IEEE Work Conference Reengineering (WCRE)*, Stuttgart, Germany, 2001.

Tibco Extensibility. "Turbo XML," **http://www.tibco.com**, current as of July 7 2001.

WWW Consortium. "XML, XML Schema, DOM," **http://www.w3.org/**, current as of March 3, 2002.

Waser, M. "Posting Development Information in IDML Format," Draft Working Paper, Washington DC: World Bank, **http://www.idmlinitiative.org/**, current as of current March 1, 2001.

XML:DB Initiative. "Questions," **http://www.xmldb.org/faqs.html**, current as of March 3, 2002.

ABOUT THE AUTHOR

Stefan Huesemann is a research and teaching assistant at the Department of Informatics at the University of Fribourg, Switzerland. Since 1999, he has taught various database and information management exercises. His research has specialized in Data Warehouses and XML. In his doctoral work, he is doing research on information systems for humanitarian organizations. His focus is on the use of XML for information exchange. He graduated from the University of Fribourg and has a bilingual "licence" in Business Administration with a specialization in financial management. He worked as an intern in the IS departments of several multinational and mid cap companies. Visit Stefan's home page at **www.stefan.huesemann.com**.

APPENDIX

Figure A1 is a graphical representation of the complete IDML schema (version 0.91).

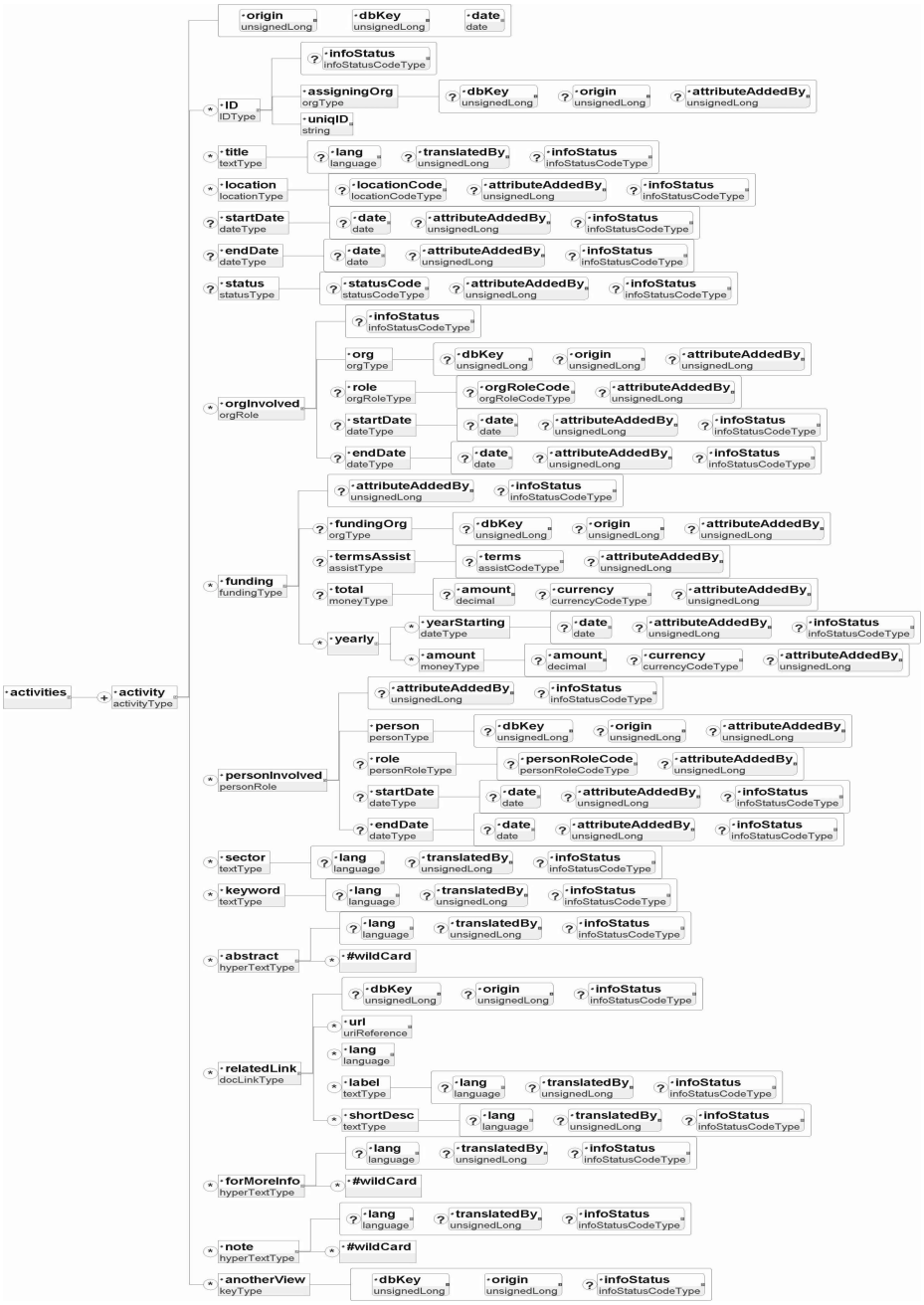

Figure A1. Hierarchical Diagram of IDML Schema

Figure A2 shows the full relationships diagram of AIDA PC.

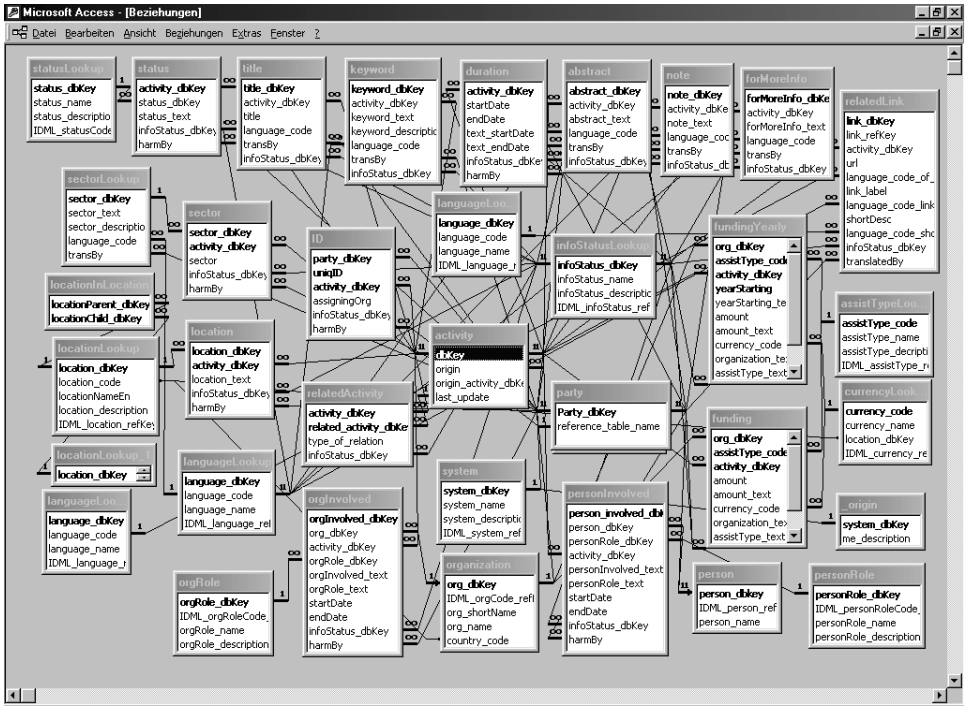

Figure A2. Relationships Diagram of AIDA PC

Journal of the Association for Information Systems
ISSN: 1536-9323