



A Conceptual Schema Approach for XML Schema Design

by

Ms. Ovilliani Yenty Yuliana

A Thesis of the Twelve-Credit Course
CS 7000 Master Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer Information Systems
Assumption University

November 2004

A Conceptual Schema Approach for XML Schema Design

by
Ms. Oviliani Yenty Yuliana

A Thesis of the Twelve-Credit Course
CS 7000 Master Thesis


Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer Information Systems
Assumption University

November 2004


Thesis Title A Conceptual Schema Approach for XML Schema Design
Name Ms. Oviliani Yenty Yuliana
Thesis Advisor Assoc.Prof.Dr. Suphamit Chittayasothorn
Academic Year November 2004

The Graduate School of Assumption University has approved this final report of the twelve-credit course, CS 7000 Master Thesis, submitted in partial fulfillment of the requirements for the degree of Mater of Science in Computer Information Systems.


Approval Committee:




(Assoc.Prof.Dr. Suphamit Chittayasothorn)
Advisor



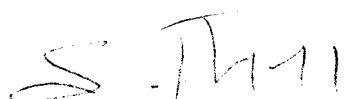
(Prof.Dr. Srisaksi Charmonman)
Chairman



(Air Marshal Dr. Chulit Meesajjee)
Dean and Co-advisor



(Asst.Prof.Dr. Vichit Avatchanakorn)
Member



(Assoc.Prof.Somchai Thayarnyong)
CHE Representative

November 2004

ABSTRACT

eXtensible Markup Language (XML) is certainly a hot topic these days because XML is rapidly becoming the premier method for exchanging and transferring information across the Internet. The studied of Giga Information Group shows the highest percentages XML is used in communication and integration. For this purposes Flat XML is addressed. On the other hand, for presentation data, Hierarchical XML is used. Another survey shows the usage of XML has been on the upswing. Nevertheless, the quality of XML Schema conceptual design frequently is still problem. In general the higher XML Schemas is, the more difficult to maintain. Software tool with a good conceptual schema approach is needed. Therefore, the research in conceptual approach for XML is becoming increasingly relevant.

This research project emphasizes on the XML file transfer. The objectives of the project are to develop a technique reengineering poorly designed XML schemas into well-normalized ones and to implement the technique in a software tool. The inputs to the tool are well-formatted Flat XML Schemas and well-formatted and validated Flat XML Document. The output of the tool is the optimal normal form of Flat XML, as a main concern of the research, which guarantees not to have redundancies.

The Nijssen's Information Analysis Methodology (NIAM) is used to represent the XML conceptual schema. The study describes how the Hierarchical XML is transformed into Flat XML. Moreover, the input Flat XML Schemas is reversed engineer into the corresponding NIAM conceptual schema using information obtained from both XML schemas and XML documents. The forward engineering from the NIAM schema into well-defined XML Schemas and the generation of corresponding XML Documents are also discussed.

ACKNOWLEDGEMENT

The writer acknowledges with profound gratitude a great many people who have had constant support and belief in her before and during her studies and her research.

She would like to thank to her husband and her parent for giving her an endless love. Their spirit of love makes her able to complete the study.

She is also thankful to Association of Christian University and Colleges in Asia, Assumption University, and Petra Christian University for mental and financial support and cooperation during her MS CIS program.

She wishes to express her deep appreciation to her thesis advisor, Assoc. Prof. Dr. Suphamit Chittayasothorn, for his continuous assistance, guidance, encouragement, patient, and support throughout the master program. In addition, he has opened her mind, inspired, and gave her the best experience to conduct research, to publish paper in the right way, and to put student in the forefront.

The writer is indebted to the qualifying examination committee: Prof. Dr. Srisaksi Charmonman, Assoc. Prof. Somchai Thayarnyong, Air Marshal Dr. Chulit Meesajjee, and Asst. Prof. Dr. Vichit Avatchanakorn for their valuable time in providing her with constructive comments and advice throughout the research.

Finally, she thanks also to her Greater Grace Church friends and Bethany International Church Bangkok friends for their support in pray. Furthermore, Mr. Sanga Rujipongpai, Mr. Rangsan Traibutra, Ms. Thanyaporn Tansatian, Ms. Myint Myint Sein, and the Graduate Office staff for their assistance her during the study in MS CIS program.

TABLE OF CONTENT

<u>Chapter</u>	<u>Page</u>
ABSTRACT	i
ACKNOWLEDGEMENTS	ii
LIST OF FIGURES	vi
LIST OF TABLES	xi
I. INTRODUCTION	1
1.1 Background of the Thesis	1
1.2 Objectives of the Thesis	4
1.3 Scope of the Thesis	4
1.4 The Professional Significance of the Thesis	6
1.5 Limitations and Assumption of the Thesis	7
1.6 Thesis Plan	7
II. LITERATURE REVIEW	10
2.1 The Hierarchical Data Model	11
2.2 eXtensible Markup Language (XML)	18
2.3 Nijssen's Information Analysis Methodology (NIAM)	40
2.4 Visual Basic (VB).NET and ActiveX Data Objects (ADO).NET	52
2.5 The Sustaining Empirical Literature	61
2.6 The Essential Empirical Literature	64
2.7 Summary	68
III. RESEARCH METHODOLOGY	70
3.1 Determine and Define the Thesis Theme	70
3.2 Research Method	70

<u>Chapter</u>	<u>Page</u>
3.3 Research Design	70
IV. SYSTEM DEVELOPMENT	73
4.1 Comparison Hierarchical Data Model and Hierarchical XML	73
4.2 Comparison NIAM Conceptual Schema and XML Conceptual Schema	82
4.3 The NIAM Conceptual Metaschema	83
4.4 The Transforming from Hierarchical XML into Flat XML Algorithms	88
4.5 The Reverse Engineering from XML Schemas into NIAM Conceptual Schemas Algorithms	98
4.6 The Forward Engineering from NIAM Conceptual Schemas into XML Schemas Algorithms	105
4.7 Input Design and Output Design Software Tool	123
V. SYSTEM EVALUATION	131
5.1 Evaluate to XML Editors	131
5.2 Evaluate to VB.NET and ADO.NET	131
5.3 Evaluate to Algorithm	133
5.4 Evaluation to the Created XML Schemas and XML Documents	134
VI. CONCLUSION AND RECOMMENDATIONS	137
6.1 Conclusions	137
6.2 Recommendations	139
6.3 Suggestion for Further Research	140
APPENDIX A XML SCHEMA AND XML DOCUMENT INPUT	141
APPENDIX B XML SCHEMA AND XML DOCUMENT OUTPUT	156
APPENDIX C CHECKED WELL-XML SCHEMA OUTPUT AND CHECKED WELL-FORMED AND VALIDATED XML DOCUMENT OUTPUT	165

<u>Chapter</u>	<u>Page</u>
APPENDIX D THE META TABLES	167
BIBLIOGRAPHY	170



LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1.1 The Thesis Schedule	9
2.1 The Suppliers and Parts Database Instances	11
2.2 A Hierarchical Diagram for Part of the COMPANY Database	13
2.3 Tree Representation of the Hierarchical Schema Figure in 2.2	13
2.4 Occurrences of PCR Types	14
2.5 Representing a Many-to-Many Relationship	16
2.6 A Hierarchical Occurrence of the Hierarchical Schema Figure 2.2.	16
2.7 Hierarchical Sequence for the Occurrence Tree in Figure 2.6.	18
2.8 The Related SGML, HTML, XML, and XHTML	19
2.9 The Listing of One Mortgage XML Document Instance	21
2.10 The Listing of the XML Schema Declaration in XML Document	23
2.11 The Listing of the Complex Type recordType Definition in Mortgage XML Schema	27
2.12 The Listing of the Complex Type documentType Definition in Mortgage XML schema	28
2.13 The Listing of Keys and Keys' Reference Definition in MortgageNew XML Schema	32
2.14 A Visualization Hierarchical Schema for Mortgage XML Schema	34
2.15 The Listing of the myInteger Range 10000-99999 Definition	36
2.16 The Listing of the Simple Type "loanNumberType" Definition	37
2.17 The Listing of the Enumeration Facet Usage	38
2.18 A Visualization Flat Schema for MortgageNew XML Schema	39
2.19 The Listing of a Comparison between the Hierarchical XML Schema and the Flat XML Schema	41

<u>Figure</u>	<u>Page</u>
2.20 A Reference Type in NIAM	44
2.21 Suppliers and Parts Knowledge Base Diagram	46
2.22 A Quaternary Fact Type That is Splitable (FD Added)	49
2.23 Default Procedure for Mapping One-to-One Fact Types	51
2.24 Suppliers and Parts Relational Mapping from NIAM Conceptual Schema into Relation Schema	52
2.25 Suppliers and City Relational Mapping for One-to-One Relationship	53
2.26 The Disconnected ADO.NET Objects Hierarchy	54
2.27 A Entity Type SUPPLIER, Its Value Types and Its Corresponding XML Schema	65
2.28 A NIAM Schema with Many-to-one Relationships and a Corresponding XML Schema	66
2.29 A NIAM Schema with Many-to-many Relationships and a Corresponding XML Schema	66
2.30 A NIAM Schema with a Ternary Fact Type and a Corresponding XML Schema	67
3.1 The Conceptual Framework	71
4.1 Hierarchical Mortgage XML Schema in ADO.NET	74
4.2 Hierarchical Mortgage XML Document in ADO.NET	75
4.3 Hierarchical Supplier-Part-A XML Schema in ADO.NET	76
4.4 Hierarchical Supplier-Part-A XML Document in ADO.NET	76
4.5 Implemented PCR type in Mortgage XML Schema (documentType Complex Type)	79
4.6 Comparison the Many-to-Many Relationship in Hierarchical Schema	80
4.7 Implementation Many-to-Many Relationship in Supplier-Part-A Hierarchical XML Schema	81
4.8 The NIAM Conceptual Metaschema	86

<u>Figure</u>	<u>Page</u>
4.9 The Meta Tables	87
4.10 Transforming Hierarchical XML Schemas into Flat XML Schemas Block Diagram	90
4.11 Flat XML Schema Result in DocumentDS	93
4.12 Reference Table for Transformation Hierarchical XML into Flat XML	94
4.13 Converting Hierarchical XML Documents into Flat XML Documents Block Diagram	96
4.14 Flat Mortgage XML Document	97
4.15 Flat Supplier-Part XML Document in ADO.NET for Figure 4.6. (a)	99
4.16 Flat Supplier-Part XML Document in ADO.NET for Figure 4.6. (c)	100
4.17 Reverse Engineering Block Diagram	101
4.18 End State Meta Tables (Part of Appendix D)	102
4.19 Supplier Part Conceptual Schema Diagram	103
4.20 Forward Engineering Block Diagram	106
4.21 Grouping the Elementary Fact Types in Meta Tables by the Main Entity Type	109
4.22 Mapping NIAM Schemas in Meta Tables into Relational Schemas	110
4.23 Grouping Fact Type for the Single Uniqueness Constraints	112
4.24 Grouping Fact Type for the Compound Uniqueness Constraints	113
4.25 Mapping One-to-One Relationship for Supplier and City Meta Table into Relational Schema	114
4.26 Conversion Data to New Relation Schemas	115
4.27 Filled Normalize the Fifth Normal Form Supplier Part Tables	116
4.28 Writing Flat XML Schema Block Diagram	117
4.29 Defining KeyRef in XML Schema	119

<u>Figure</u>	<u>Page</u>
4.30 Defining Supplier Table Schema as a Supplier Complex Type	120
4.31 Defining Key in XML Schema	121
4.32 Writing Flat XML Document Block Diagram	122
4.33 Defining the Instances Supplier Table in XML Document	124
4.34 Main Menu	125
4.35 Analysis Sub Menu	126
4.36 Show Sub Menu	126
4.37 XSD Open File Dialog	127
4.38 Transform the Hierarchical to the Flat XML	127
4.39 XML Schema Information	128
4.40 NIAM Conceptual Schema	128
4.41 Confirmation the Deleting Fact Type	129
4.42 Modification and Confirmation Fact Type Information	129
4.43 Displaying the Original XML File	130
4.44 Displaying the Produced XML File	130
5.1 Visualization the Right MortgageNew XML Schema	134
5.2 Visualization the Wrong MortgageNew XML Schema	134
A.1 The Listing of the Mortgage XML Schema	141
A.2 The Listing of the Mortgage XML Document	142
A.3 The Listing of the Supplier-Part XML Schema	144
A.4 The Listing of the Supplier-Part XML Document	146
A.5 The Listing of the Supplier-Part-A XML Schema	149
A.6 The Listing of the Supplier-Part-A XML Document	150
A.7 The Listing of the Supplier-Part-C XML Schema	152

<u>Figure</u>	<u>Page</u>
A.8 The Listing of the Supplier-Part-C XML Document	153
B.1 The Listing of the MortgageNew XML Schema	156
B.2 The Listing of the MortgageNew XML Document	158
B.3 The Listing of the Supplier-PartNew XML Schema	160
B.4 The Listing of the Supplier-PartNew XML Document	162
C.1 Checked Well-Formed MortgageNew XML Schema Output	165
C.2 Checked Well-Formed MortgageNew XML Document Output	165
C.3 Checked Validated MortgageNew XML Document Output	166
C.4 Checked Well-Formed Supplier-PartNew XML Schema Output	166
C.5 Checked Well-Formed Supplier-PartNew XML Document Output	167
C.6 Checked Validated Supplier-PartNew XML Document Output	167
D.1 The Populated Supplier Part Meta Tables	167

LIST OF TABLES

<u>Table</u>	<u>Page</u>
2.1 Attribute of the <xsd:element> Tag	24
2.2 Datatypes to which Each Facet Applies	25
2.3 Product Middleware	35
2.4 XML-Enabled Databases	36
2.5 Some Basic Symbols Used in Conceptual Schema Diagram	44
2.6 Key Length Check for Ternaries and Longer Fact Types	49
2.7 Properties of the DataColumnn Object	57
4.1 Implemented XML Attributes in DataColumnn Properties ADO.NET	84
4.2 Correctness Suppliers and Parts Functional Dependency	106
5.1 The Comparison XML File	136

I. INTRODUCTION

1.1 Background of the Thesis

The emergence of the Internet creates a big challenge for research, education, and scientific communities to share information and do research. In addition, the Internet opens up in recent years to more commercial uses. With the Internet, a dedicated connection for exchanging and transferring data is no longer required – providing both parties are on the Internet. To support exchanging or transferring data within the Internet, World Wide Web Consortium (W3C) recommended eXtensible Markup Language (XML). Nevertheless, the quality of conceptual design of XML Schema that will be transferred frequently is still poor.

XML is certainly a hot topic in the software community these days because XML is rapidly becoming the premier method for exchanging and transferring information across the Internet. Nonetheless, at this point of time XML is also being used to simply exchanging data and integration data between different platforms and applications. Because XML is a text-based format therefore it can easily be moved across platforms and can be moved around the existing Internet technologies and protocols. On the other hand, traditional software integration is difficult to exchange data between platforms.

A study about XML usage among companies is done by Giga Information Group in 2001 (Daum 2003). The study shows that XML is used in different areas. XML was used for: data exchange and messaging, 33%; application integration, 27%; data integration, 13%; content publishing, 12%; the construction of portals, 6%; other purposes, 6%. Not surprisingly most areas are somehow connected to communication and integration (the three highest percentages). Another survey is done by Silicon

Valley research firm (Pastore 2001). They say the usage of XML has been on the upswing, nearly doubling in the last six months. On Dec 9, 2000, developers used XML 25.7%, according to the survey. Six month latter, on May 9, 2001, the number increases to 38.2 %. Finally, developers plan to use XML, 53.1 % in 2002.

As the number of XML usage is on the upswing, the number of XML schema also is grow. If XML Schemas are still in a poor design then maintenance to these XML Schemas also increase as high as the usage of XML. Maintenance requires understanding and effort because XML Schema is textual and is described using XML syntax, this becomes tedious. In general, the larger the XML Schemas need the more effort to understand and maintain. Software tool with a good conceptual schema approach is needed. As a result, research in the area conceptual approach for XML is becoming increasingly relevant.

The most of the problem with many database applications can be traced to a bad database design (Halpin 1995) because most people only talk about the third normal form (Becker 1998). Moreover several authors of Systems Analysis and Design text book state that normalization on conceptual schema design enough until the third normal form (Dennis 2002) (Hoffer 2002) (Whitten 2004). However, Date (2000) and Halpin (1995) mention that in the third normal form conceptual schema still possible is not in Boyce/Codd Normal Form (BCNF). In addition the conceptual schema is still found Multi Value Dependency (MVD, no full fill the fourth normal form) and the conceptual schema is still split able (Join Dependency, JD, no full fill the fifth normal form). Therefore database schema is in a bad design and the redundancy data can not be avoided. The impacts of redundancy data are certain update anomaly and inconsistency data. As a solution, Halpin suggested the rich constraints and intuitive modeling techniques, Nijssen's Information Analysis Methodology (NIAM).

The other biggest problem is the human factor. Today, software engineers in the field of XML come from three directions: the SGML camp, the object-oriented camp, and the relational camp. Members of the SGML camp, for example, who are used to a more document-centric design style, will have to adapt to the more data-centric style. They will also find the concepts such as entity relationship modeling and referential integrity are exciting new fields where there remains a lot to do. Members of the SQL camp, in contrast, will miss concepts of referential integrity in XML but will find that the rich structuring possibilities that exist in XML open a whole new world of database design. Finally members of the object-oriented camp will be sad because they miss a behavior model in XML documents. On the other hand, they may find it exciting that XML actually does make remote procedure calls work across company, platform, and language boundaries.

To fulfill the human factor, there are two structures of XML, i.e. a Hierarchy XML and a Flat XML. The Hierarchical XML is used to support HTML for presentation data with duplication data in sub element. The Flat XML, however, is used for transportation data. The survey above shows that the biggest percentage XML is used for communication and integration. At the present time mostly companies apply relational database application. Therefore, Flat XML is useful for transferring data between two relational databases. Furthermore, Elmasri (1994) said that in general the hierarchical model works well for database applications that are naturally hierarchical. However, when there are many nonhierarchical relationships, trying to fit those relationships into a hierarchical form is difficult. Also the results are often unsatisfactory. As a result, while users transfer Hierarchical XML, Software tool must transform it to Flat one first.

The biggest question in the researcher mind is “Can the NIAM conceptual schema, the Conceptual Schema Design Procedure (CSDP) methodology and the Relational mapping (Rmap) procedure, is applied meaningfully to improve the poorly designed XML Schema?”

1.2 Objectives of the Thesis

The objectives that want to reach with this research are

- (1) To find a good conceptual modeling technique for XML Schemas design.

In order to improve poorly designed XML schemas to be better ones.

- (2) To generate a software tool for reengineering XML Schemas using the conceptual schema approach.

1.3 Scope of the Thesis

The scope of the thesis comes into view on conceptual framework in Figure 3.1. The overall objective of the thesis is to improve poorly designed XML schemas to be better ones. The inputs to the creating software tools are well-formed XML Schemas and XML Document and validated XML Documents against XML Schemas. Therefore the creating software tools do not create a sub program for producing XML Schemas and XML Documents form database and vice versa. Moreover the designing software tools do not create a sub program for checking well-formed and validated XML. In addition, because XML Documents is validated by XML Schemas, the software tool reengineers XML Schemas. If the inputs are Hierarchical XML then the software tool transforms the inputs into the Flat one first. The input Hierarchical XML in this study is the hierarchical for transforming data, therefore the sub element is directly to the main element (see Figure 4.5.).

To improve a poorly designed XML schemas researcher apply NIAM conceptual schema approach. The NIAM conceptual schema consists of three main sections, i.e.:

stored fact types, constraints, and derivation rules. The designing software tool applies fully stored fact types. However, not all constraints apply in the software tool because of the limitation of ActiveX Data Object (ADO).NET and time to study the other XML method. The limitation ADO.NET will be discussed in section 4.2. Constraints that will implement in the software tool are validation rules or integrity rules (uniqueness constraint and reference type) and restriction to apply population (maxOccurs and minOccurs). The other constraints, such as lists various constraints and arithmetic derivations will not implement. W3C still not define function, operators and rules that may be used to derive information to support arithmetic derivations. Therefore derivation rules will not be implemented in the software tool. However, researcher designs meta tables to capture all information about the stored fact types and the constraints. Researcher uses Visual Basic (VB).NET and ADO.NET to develop the software tool.

To reverse engineering XML Schemas into NIAM conceptual schemas the software tool employs the Conceptual Schema Design Procedure (CSDP) methodology. The NIAM conceptual schemas are stored in meta tables. The software tool starts from the fourth step CSDP (check uniqueness constraints and arity of fact types). If the fact type (complex type) is splittable, it automatically does the first (transforming familiar information examples into elementary facts) and the second (applying a population check) steps of CSDP, followed with the fifth (add mandatory role constraints) and the seventh (perform final checks) steps of CSDP. The software tool requests participation universe of discourse (UoD) expert to complete or modify fact type, uniqueness constraint, and mandatory constraint. In this study, the CSDP still not addressed are the third and the sixth steps because of the limitation mentioned above.

To forward engineering NIAM conceptual schemas in the meta tables into relational schema the software tool applies Relational Map (Rmap) procedure. After that the software tool creates XML Schemas from relational table schema. Finally the software tool converts XML Documents into the other XML Documents with the new XML Schemas. For this sub program, the researcher uses XML method WriteXmlSchema and WriteXml. WriteXmlSchema is used to write XML Schemas and WriteXml is used to write XML Document.

The output of the software tool is file and screen. The researcher does not design output to hard copy. In addition, the file size outputs are not the first main concern in this research. The first priorities are to find a good conceptual modeling technique for XML Schema and to generate a software tool for reengineering XML Schemas using the conceptual schema approach, as mentioned in the objectives of thesis. In other word to improve the poorly designed XML Schema to become the fifth normal form.

1.4 The Professional Significance of the Thesis

The researcher points out in the background of thesis that the usage of XML on the upswing. As a result, study in the area conceptual approach for XML is becoming increasingly relevant. It is expected that the conducted research of “**A Conceptual Schema Approach for XML Schema Design**” will contribute the well conceptual modeling technique for XML Schemas design to improve the normalized XML Schemas. Therefore, the produced XML Schemas is not only the well formatted and the validated XML Schemas but also the normalized XML Schemas. As far as the researcher knows, no the other researchers do it. In addition, the researcher wants to enrich the existing XML Editors in the market that are not just only editing and checking well-formed and validated but also normalizing the XML Schemas.

1.5 Limitations and Assumption of the Thesis

To conduct the study, researcher assumes that

- (1) Available database application and program application have a feature to transform from database to XML Documents and XML Schemas and vice versa.
- (2) The inputs of designing software tools are in the well-formed and the validated XML Documents and in the well-formed XML Schemas.
- (3) The name of XML Documents is similar to the name of XML Schemas.
- (4) XML Documents are significant as a population of elementary facts checking.
- (5) If maxOccurs does not exit in **Attribute-Facet** Table it means the maxOccurs is one.

There are limitations that researcher discovers while conducting the research, such as:

- (1) No derivation rules in XML, such as arithmetic and function, therefore checking derivation rules are not implemented in software tool.
- (2) ADO.NET. can not capture all attributes and data facets XML Schema.
- (3) VB.NET does not support attribute xsi:schemaLocation in order to connect XML documents with XML schema.
- (4) No attribute to define **msdata:IsDataSet="true"**, therefore the XML Schema output was still in Hierarchical XML if XML Schema were written by **XmlTextWriter** method.

1.6 Thesis Plan

To keep the research on time, the researcher makes a research schedule as shown in Figure 1.1. The researcher divides the study into four categories of tasks, i.e. thesis

preparation, analysis and design software tool, implementation software tool, and thesis report.

In thesis preparation task, researcher reviews literature to define the topic. After that the researcher studies intensively about NIAM, XML, VB.NET, and ADO.NET. Furthermore, the researcher writes statement of problem, proposal of the study, research question, professional significant, literature review, and research methodology in proposal. Next, the proposal is submitted. Finally, the researcher defends the proposal.

In analysis and design software tool task, the researcher analyzes the Hierarchical Data Mode and Hierarchical XML, the NIAM conceptual schema and XML conceptual schema. After that the researcher designs the NIAM conceptual metaschema. Based on the knowledge of analysis, the researcher designs the transforming Hierarchical XML into Flat XML algorithm. Then the researcher creates the reverse engineering from the XML Schema into the NIAM conceptual schema algorithm and creates the forward engineering from the NIAM conceptual schema into the XML Schema algorithm, and finally designs input and output software tool.

In implementation software tool task, the researcher realizes all algorithms that create in the previous task. After each program is almost complete, the researcher tests and validates the program. Finally, the researcher simulates the software tools with several XML Schemas and XML Documents.

In the last task, the researcher writes the thesis report and submits it for checking grammar, as well as edits the thesis report. Finally, the researcher defends the thesis.

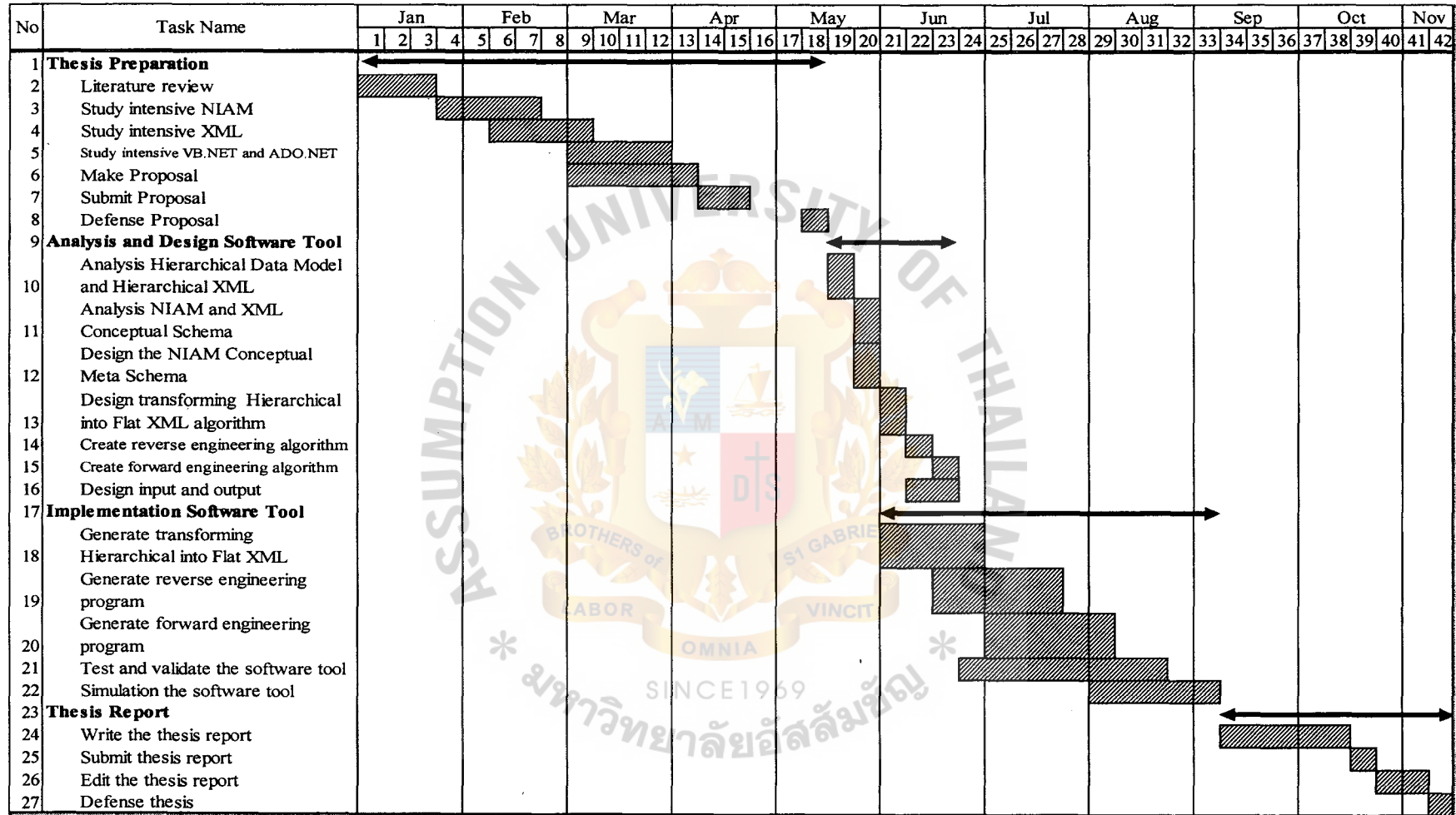


Figure 1.1. The Thesis Schedule.

II. LITERATURE REVIEW

This chapter reviews the theoretical literatures and the empirical literatures for supporting the conceptual framework research that will be conducted. The theoretical literatures spotlight on the following subject areas hierarchical data model briefly, Nijssen's Information Analysis Methodology (NIAM) conceptual schema, and overview of XML Schema and XML document. Moreover several tools and application programs that will be used in the study, such as XML Writer, Microsoft's Visual Studio .NET, ActiveX Data Objects and Visual Basic .NET, will be discussed shortly. In addition the empirical literatures focus on the prior researches that related with conducted research will be reviewed. Researcher separates the empirical literatures into two categories. One is the sustaining empirical literature to show what researches have already been conducted and what are still open. The others are the essential empirical literature that will be adopted to conduct the research with modification.

In study, the researcher uses two examples. The first example is a popular the Suppliers and Parts Database (Date 2000). The researcher modifies the status “30” to “10” for “S3”. Figure 2.1. shows many-to-many relationship between Suppliers and Parts. The second example is a comprehensive enough Mortgage Information XML Schema and XML Document (Holzner 2004) with modification, hierarchical XML is used to transform data, therefore the Mortgage sub-element is directly to the Document main-element (see Figure A.1. and Figure A.2.). The example shows one-to-many relationship between Document and Mortgage and many-to-one relationship between Document and Mortgagee and between Document and Bank. The researcher uses another example in hierarchical data model because there are several similarities and differences between the theoretical hierarchical data model and hierarchical XML.

41035 e.1

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	10	Paris
S4	Clark	20	London
S5	Adams	30	Athens

(a) Suppliers Table

P#	PNAME	COLOR	CITY
P1	Nut	Red	London
P2	Bolt	Green	Paris
P3	Screw	Blue	Rome
P4	Screw	Red	London
P5	Cam	Blue	Paris
P6	Cog	Red	London

(b) Parts Table

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

(c) Suppliers-Parts Table

Figure 2.1. The Suppliers and Parts Database Instances.

2.1 The Hierarchical Data Model

In this section, the principles behind of the hierarchical model (Elmasri 1994) are discussed. Firstly, parent-child relationships and how they can be used to form a hierarchical schema are discussed. Follow by properties of hierarchical schema and occurrence trees. Lastly, the hierarchical occurrence trees and the common method for storing the trees are discussed.

(1) Parent-Child Relationships and Hierarchical Schemas

The hierarchical model represents data by emphasizing hierarchical relationships. The main structures used by the model are **record types** and **Parent-Child Relationship** (PCR) types. Records of the same type are grouped into **record types**. A record type is given a name, and its structure is defined by a collection of named **fields** or **data items**. Each field has a certain data type, such as integer, real, or string.

A PCR type defines a hierarchical **one-to-many** relationship between a **parent record type** and a **child record type**. The record type on the one-side is called the parent record type of the PCR type, and the one on the many-side is called the **child record type** of the PCR type. An **occurrence** of the PCR type consists of one record of the parent record type and a number of records (zero or more) of the child record type. Relationships are strictly hierarchical in that a record type can participate as child in at most one PCR type. This restriction makes it difficult to represent a database where numerous relationships exist.

The **hierarchical database schema** basically is a tree data structure. Figure 2.2. shows a hierarchical diagram for a hierarchical schema with six record types and five PCR types. The record types are DEPARTEMENT, EMPLOYEE, PROJECT, DEPENDENT, SUPERVISEE, and WORKER. Field names can be displayed under each record type name. In brevity, the diagram displays only the record type names. Corresponding to a hierarchical schema, a number of occurrence trees will exist in the database (see Figure 2.3.). Researcher uses the hierarchical diagram to visualize XML Schema with Microsoft Visual Studio .NET. and tree representation will be used to simplify the schema.

A PCR type in a hierarchical schema is referred by listing the pair (parent record type, child record type) between parentheses, for example (DEPARTEMENT, EMPLOYEE) and (DEPARTEMENT, PROJECT). Each occurrence of the (DEPARTEMENT, EMPLOYEE) PCR type relates one department record to the records of the many (zero or more) employees who work in that department. An occurrence of the (DEPARTEMENT,

PROJECT) PCR type relates a department record to the records of projects controlled by that department. Figure 2.4. shows the two PCR occurrences (or instances) example for each of these two PCR types.

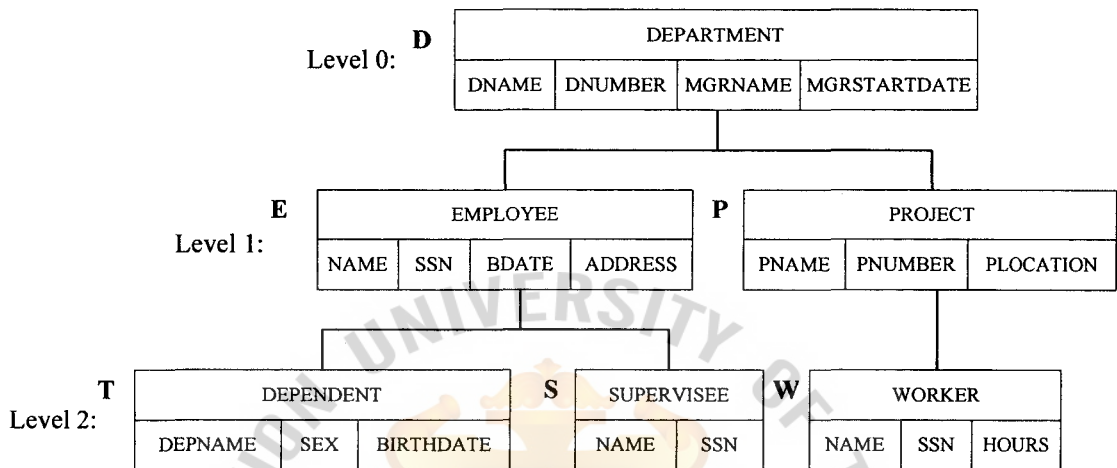


Figure 2.2. A Hierarchical Diagram for Part of the COMPANY Database.

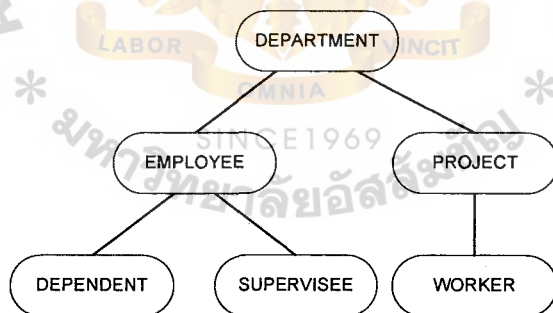


Figure 2.3. Tree Representation of the Hierarchical Schema Figure in 2.2.

(2) Properties a Hierarchical Schema

The hierarchical schemas of record type and PCR type need the following properties:

- (a) One record type, called the root of the hierarchical schema, does not participate as a child record type in any PCR type.
- (b) Every record type except the root participates as a child record type in exactly one PCR type.
- (c) A record type can participate as parent record type in any number (zero or more) of PCR types.
- (d) A record type that does not participate as parent record type in any PCR type is, called a leaf of hierarchical schema.
- (e) If a record type participates as parent in more than one PCR type, then its child types are ordered. The order is displayed, by convention, from left to right in a hierarchical diagram.

DEPARTMENT:

EMPLOYEE:



- (a) Two occurrences of the PCR type (DEPARTEMENT, EMPLOYEE)

DEPARTMENT:

PROJECT:



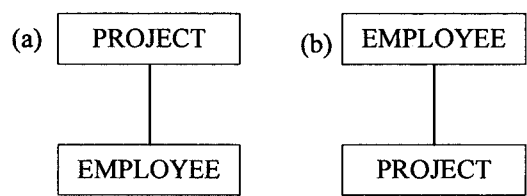
- (b) Two occurrences of the PCR type (DEPARTEMENT, PROJECT)

Figure 2.4. Occurrences of PCR Types.

The preceding properties of a hierarchical schema mean that every node, except the root has exactly one parent node. However, a node can have several child nodes, and in this case they are ordered from left to right. In Figure 2.2., EMPLOYEE is the first child of DEPARTEMENT, and PROJECT is the second child. The identified properties also limit the types of relationships that can be represented in a hierarchical schema. In particular, many-to-many relationships between record types cannot be directly represented, because parent-child relationships are one-to-many relationship, and a record type cannot participate as child in two or more distinct parent-child relationships.

Elmasri proposes duplication of child record instance or Virtual Parent-Child Relationship to handle a many-to-many relationship in the hierarchical model. In the study, because XML implement duplication in sub-element, this section will discuss shortly the first propose. For example, consider a many-to-many relationship between EMPLOYEE and PROJECT, where a project can have several employees working on it, and an employee can work on several projects. The relationship PROJECT and EMPLOYEE as PCR type is shown in Figure 2.5. (a). In this case a record describing the same employee can be duplicated by appearing once under each project that the employee works for. Alternatively, relationship EMPLOYEE and PROJECT, see Figure 2.5. (b), in which case project records may be duplicated. For example, consider the instances of the EMPLOYEE: PROJECT in Figure 2.6. These instances are stored using the hierarchical schema of Figure 2.5. (a). There are two occurrences of the (PROJECT, EMPLOYEE) PCR for each project. The employee records for

Zelaya and Jabbar will appear twice each as child records, however, because each of these employees works on two projects, Computerization and Newbenefits.



(a) One Representation of the Many-to-Many Relationship

(b) Alternative Representation of the Many-to-Many Relationship

Figure 2.5. Representing a Many-to-Many Relationship.

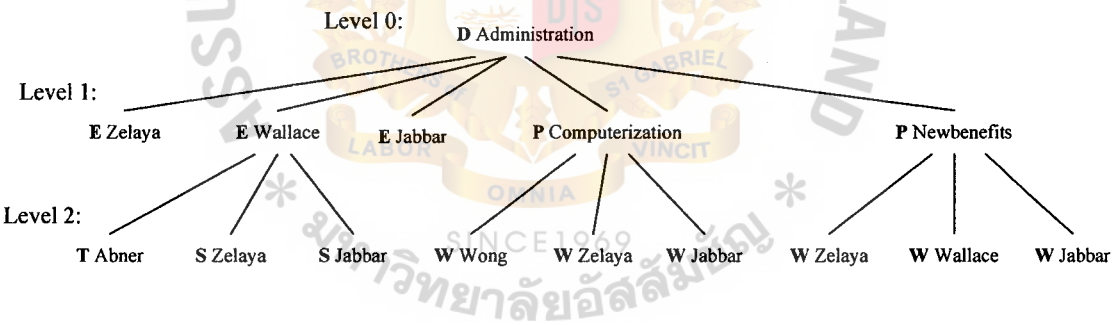


Figure 2.6. A Hierarchical Occurrence of the Hierarchical Schema Figure 2.2.

(3) Hierarchical Occurrence Trees

Each hierarchical occurrence, also called an occurrence tree, is a tree structure whose root is a single record from the root record type. The occurrence tree also contains all the children record occurrences of the root record, all children record occurrences within the PCRs of each of the child

records of the root record, and so on, all the way to records of the leaf record types.

For example, consider the hierarchical diagram shown in Figure 2.2. In the occurrence tree, each node is a record occurrence, and each arc represents a parent-child relationship between two records. In Figure 2.2. and Figure 2.6., character **D**, **E**, **P**, **T**, **S**, and **W** is used to represent **type indicators** for the record types DEPARTMENT, EMPLOYEE, PROJECT, DEPENDENT, SUPERVISEE, and WORKER, respectively. The indicators will see significantly in hierarchical sequences (see section 2.1 (4)).

(4) Linearized Form of a Hierarchical Occurrence

A hierarchical occurrence tree can be represented in storage by using any of a variety of data structures. However, a particularly simple storage structure that can be used is the **hierarchical record**, which is a linear ordering of the records in an occurrence tree in the preorder traversal of the tree. This order produces a sequence of record occurrence tree known as the **hierarchical sequence** (or **hierarchical record sequence**) of the occurrence tree. The hierarchical sequence is shown in Figure 2.7. If hierarchical sequence is used to implement occurrence trees, a record type indicator with each record is needed to be stored because of the different record types and the variable number of child records in each parent-child relationship. The system needs to examine the type of each record as it goes sequentially through the records. The hierarchical sequence is important for hierarchical data manipulation, such as in ADO.NET.

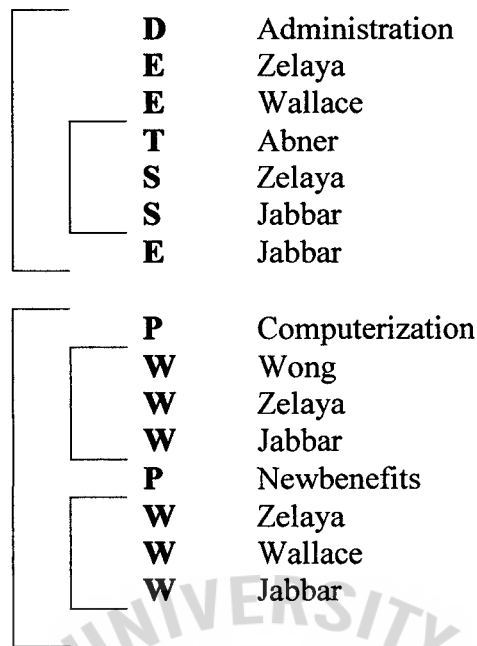


Figure 2.7. Hierarchical Sequence for the Occurrence Tree in Figure 2.6.

2.2 eXtensible Markup Language (XML)

This section covers the important mechanisms of XML Schema. It describes how to declare the elements and attribute that appear in XML documents, the distinctions between simple and complex types, defining complex types, the use of simple type for element and attribute values, schema annotation, a simple mechanism for reusing element and attribute definitions. Moreover, a mechanism for specifying uniqueness among attributes and elements will be explained. Finally, Hierarchical and Flat XML structure is discussed briefly.

XML has been developed by a working group under the umbrella of the World Wide Web Consortium (W3C). To solve the problem at its root and to create an open-ended markup language that could easily accommodate future expansions and additions, the W3C created a working group to define such a markup language. Tittel (2002) says the primary goal of this merry band of technologists is to bring the kinds of

capabilities found in SGML to the Web. By February 1998, version 1.0 of the XML specification has been unleashed on the world, and the XML markup revolution begin in earnest. XML is not a replacement for HTML. XML enables the Internet industry to invent a new set of powerful tools for many purposes. XML stores and organizes the data, and HTML renders it inside browser by using a style sheet. Figure 2.8. shows a related among SGML, HTML, XML, and XHTML.

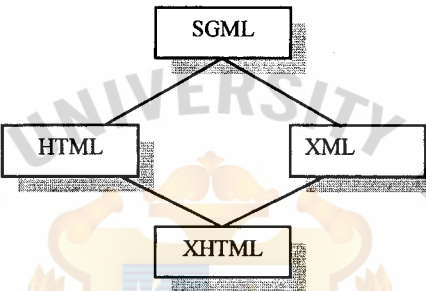


Figure 2.8. The Related SGML, HTML, XML, and XHTML.

Harold (2001) mentions that XML does not a language as the name suggests. However, it is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. XML is a meta-markup language that defines a syntax in which other field-specific markup language can be written. The detail of XML Schema primer, XML Schema structures, and XML Schema datatypes second edition can be shown respectively in Part 0 (<http://www.w3.org/TR/xmlschema-0/>), Part 1 (<http://www.w3.org/TR/xmlschema-1/>), and Part 2 (<http://www.w3.org/TR/xmlschema-2/>).

In this research, the other two important terms will be used, i.e.: valid and well-formed. **Valid** means a XML Document adhered to the rules outlined in an associated XML Schema. **Well-formed** means a XML Document or a XML Schema that adheres

to the syntax rules for XML, are explicitly designed to make document easy for a computer to interpret.

(1) XML Document

Consider an instance mortgage document in Figure 2.9., part of Figure A.1., Mortgage XML document. It describes an outstanding mortgage loan held by a real estate investor. The mortgage document consists of a main-element and sub-elements. The main-element is **document**. The sub-elements are **comment**, **mortgagee**, **mortgages**, and **bank**. Mortgages sub-elements in turn contain another sub-element, sub-sub-element **mortgage**. Figure 2.9. and Figure 2.14. show the relationship between main-element and sub-elements. Elements that contain sub-elements or carry attributes are said to have complex types, whereas elements that contain numbers, strings, dates, etc. (built-in) but do not contain any sub-elements are said to have simple types. Some elements have attributes; attributes always have simple types.

(2) XML Schema

The Document Type Definition (DTD) language, which has traditionally been the most common method for describing the structure of XML instance documents, lacks enough expressive power to properly describe highly structured data. XML Schema, on the other hand, provides a much richer set of structures, types and constraints for describing data, definitions of cardinality and is therefore expected to soon become the most common method for defining and validating highly structured XML documents. Therefore, this research conducts on XML Document with XML Schema.

```

1  <?xml version="1.0" encoding="UTF-8" ?>
    . . .
87  <document documentDate="2004-07-14">
88    <comment>Good</comment>
89    <mortgagee phone="8702205">
90      <name>Widya</name>
91      <location>Manukan</location>
92      <city>Surabaya</city>
93    </mortgagee>
94    <mortgage>
95      <loanNumber>12 3122 34</loanNumber>
96      <property>Bungalow</property>
97      <date>2004-07-12</date>
98      <loanAmount>5000</loanAmount>
99      <term>12</term>
100    </mortgage>
101    <bank phone="888.555.8888">
102      <name>XML Bank</name>
103      <location>12 Schema Place</location>
104      <city>New York</city>
105      <state>NY</state>
106    </bank>
107  </document>
108 </Root>

```

Figure 2.9. The Listing of One Mortgage XML Document Instance.

The purpose of XML Schema is to define a class of XML documents, and so the term “instance document” is often used to describe an XML document that conforms to a particular schema. For example, Mortgage XML schema (see Figure A.1.) consists of a schema element and a variety of sub-elements, most notably element, complexType, and simpleType which determine the appearance of elements and their content in instance documents.

Each of the elements in the schema has a prefix **xsd:** which is associated with the **XML Schema namespace** through the declaration, **xmlns:xsd="http://www.w3.org/2001/XMLSchema"**, that appears in the schema element. The prefix **xsd:** is used by convention to denote the XML Schema namespace, although any prefix can be used. The same prefix, and

hence the same association, also appears on the names of built-in simple types, e.g. `xsd:string`. The purpose of the association is to identify the elements and simple types as belonging to the vocabulary of the XML Schema language.

(3) Declaring the Location and the Name of XML Schema in XML Document

The association of a XML Document is a two-stage process. First, the

name	space	declaration
------	-------	-------------

`xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance` associates the namespace prefix **`xsi`** with the URI shown. Second, the **`xsi:noNamespaceSchemaLocation`** attribute, which belongs to the namespace `http://www.w3.org/2001/XMLSchema-instance` indicates the **location** and **name** of the schema. The `xsi:noNamespaceSchemaLocation` attribute can only be used when the `xsi` namespace prefix has been declared. The value of the `noNamespaceSchemaLocation` attribute indicates the location of the schema (for example see Figure 2.10.). It shows how to declare Mortgage XML Schema in Mortgage XML Document. In this example, the location of Mortgage XML Schema and Mortgage XML Document are in the same folder so it should not declare in the `noNamespaceSchemaLocation` attribute.

(4) Complex Type Definitions, Element and Attribute Declarations

In XML Schema, there is a basic difference between complex types which allow elements in their content and may carry attributes, and simple types which cannot have element content and cannot carry attributes. There is also a major distinction between definitions which create new types (both simple and complex), and declarations which enable elements and attributes

with specific names and types (both simple and complex) to appear in document instances.

```
1 <Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Mortgage.xsd">
```

Figure 2.10. The Listing of the XML Schema Declaration in XML Document.

New complex types are defined using the **complexType** element and such **definitions** typically contain a set of **element declarations**, **element references**, and **attribute declarations**. The declarations are not themselves types, but rather an association between a name and the constraints which govern the appearance of that name in documents governed by the associated schema. **Elements** are declared using the **element** element, and **attributes** are declared using the **attribute** element. Table 2.1. shows a list of the attributes that can be used in the `<xsd:element>` element. In addition, Table 2.2. gives an overview of facets and lists to which datatypes applies. Figure 2.11. shows that `recordType` is defined as a complex type, and within the definition of `recordType`. There are four element declarations and one attribute declaration.

The consequence of this definition is that any element appearing in an instance whose type is declared to be `recordType` must consist of four elements and one attribute. These elements must be called name, location, city, and state as specified by the values of the declarations' name attributes, and the elements must appear in the same sequence (order) in which they are declared. All of these elements will each contain a string.

Table 2.1. Attribute of the <xsd:element> Tag.

Attribute	Description
Abstract	Boolean value that requires the use of a substitution group
Block	Allows developer to control replacement by restriction, extension, or both derived types
Default	Default value for the element
Final	Allows developer prevent derivations by restriction, extension, or both
Fixed	A default, but unchangeable, value for the element
Form	Used to specify if the qualification of an element is to be done by a local or global declaration
Id	Unique identifier
maxOccurs	Maximum number of times the element can occur within the parent element
minOccurs	Minimum number of times the element can occurs within the parent element
Name	The name of attribute being created
Nillable	Used to specify if an element can contain a nil value (which is different than not being present)
Ref	Allows developer to reference a global element declaration, and therefore inherit some of its settings
substitutionGroup	Allows developer to assign the element to a group whereby any one element of the group can be substituted for another element instance

Table 2.1. Attribute of the <xsd:element> Tag (Continued).

Attribute	Description
type	The datatype of value of the element being created
use	Optional item that allows developer to specify whether the attribute is optional, prohibited, or required

Table 2.2. Datatypes to which Each Facet Applies.

Facet	Applies to Datatype List
enumeration	ENTITIES, ENTITY, ID, IDREF, NCName, NMTOKEN, NMTOKENS, NOTATION, Name, QName, anyURI, base64Binary, byte, date, dateTime, decimal, double, duration, float, gDay, gMonth, gMonthDay, gYear, gYearMonth, hexBinary, int, integer, language, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, normalizeString, positiveInteger, short, string, time, token, unsignedByte, unsignedInt, unsignedLong, unsignedShort
fractionDigits	byte, decimal, int, integer, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, short, unsignedByte, unsignedInt, unsignedLong, unsignedLong
length	ENTITIES, ENTITY, ID, IDREF, NCName, NMTOKEN, NMTOKENS, NOTATION, Name, QName, anyURI, base64Binary, hexBinary, language, normalizeString, string, token
maxLength	ENTITIES, ENTITY, ID, IDREF, NCName, NMTOKEN, NMTOKENS, NOTATION, Name, QName, anyURI, base64Binary, hexBinary, language, normalizeString, string, token

Table 2.2. Datatypes to which Each Facet Applies (Continued).

Facet	Applies to Datatype List
minExclusive	byte, date, dateTime, decimal, double, duration, float, gDay, gMonth, gMonthDay, gYear, gYearMonth, int, integer, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, short, time, unsignedByte, unsignedInt, unsignedLong, unsignedShort
minInclusive	byte, date, dateTime, decimal, double, duration, float, gDay, gMonth, gMonthDay, gYear, gYearMonth, int, integer, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, short, time, unsignedByte, unsignedInt, unsignedLong, unsignedShort
minLength	ENTITIES, ENTITY, ID, IDREF, NCName, NMTOKEN, NMTOKENS, NOTATION, Name, QName, anyURI, base64Binary, hexBinary, language, normalizeString, string, token
pattern	ENTITY, ID, IDREF, NCName, NMTOKEN, NOTATION, Name, QName, anyURI, base64Binary, Boolean, byte, date, dateTime, decimal, double, duration, float, gDay, gMonth, gMonthDay, gYear, gYearMonth, hexBinary, int, integer, language, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, normalizeString, positiveInteger, short, string, time, token, unsignedByte, unsignedInt, unsignedLong, unsignedShort
totalDigits	byte, decimal, int, integer, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, short, unsignedByte, unsignedInt, unsignedLong, unsignedLong

Table 2.2. Datatypes to which Each Facet Applies (Continued).

Facet	Applies to Datatype List
whiteSpace	ENTITIES, ENTITY, ID, IDREF, NCName, NMOKEN, NMOKENS, NOTATION, Name, QName, anyURI, base64Binary, boolean, byte, date, dateTime, decimal, double, duration, float, gDay, gMonth, gMonthDay, gYear, gYearMonth, hexBinary, int, integer, language, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, normalizeString, positiveInteger, short, string, time, token, unsignedByte, unsignedInt, unsignedLong, unsignedShort

```
23 <xsd:complexType name="recordType">
24   <xsd:sequence>
25     <xsd:element name="name" type="xsd:string" />
26     <xsd:element name="location" type="xsd:string" />
27     <xsd:element name="city" type="xsd:string" />
28     <xsd:element name="state" type="xsd:string" minOccurs="0"/>
29   </xsd:sequence>
30   <xsd:attribute name="phone" type="xsd:string" use="optional"
31     form="qualified" />
31 </xsd:complexType>
```

Figure 2.11. The Listing of the Complex Type recordType Definition in Mortgage XML Schema.

The recordType definition contains only declarations involving the simple types: string. In contrast (see Figure 2.12.), the **documentType** definition contains element declarations involving **complex types**, e.g. **recordType**, although note that both declarations use the same type attribute to identify the type, regardless of whether the type is **simple** or **complex**.

```

14 <xsd:complexType name="documentType">
15   <xsd:sequence>
16     <xsd:element ref="comment" minOccurs="1" />
17     <xsd:element name="mortgagee" type="recordType" />
18     <xsd:element name="mortgage" type="mortgageType"
19       minOccurs="0" maxOccurs="8" />
20     <xsd:element name="bank" type="recordType" />
21   </xsd:sequence>
22   <xsd:attribute name="documentDate" type="xsd:date" />
23 </xsd:complexType>

```

Complex
Type

Figure 2.12. The Listing of the Complex Type documentType Definition in Mortgage XML schema.

In defining documentType, two of the element declarations, for **mortgagee** and **bank**, associate **different element names** with the **same complex type**, namely **recordType**. The consequence of this definition is that any element appearing in an instance document whose type is declared to be documentType must consist of elements named mortgagee and bank, each containing the four sub elements (name, location, city, and state) that are declared as part of recordType. The documentType definition contains a documentDate attribute declaration, identifies a simple type. In fact, all attribute declarations must refer simple types because attributes cannot contain other elements or other attributes.

The element declarations described so far have associated a name with an existing type definition. Sometimes it is **preferable to use an existing element rather than declare a new element**. For example, see Figure 2.12., element **ref comment**. The value of the ref attribute must **refer** to a **global element**, i.e. one that has been declared under schema rather than as part of a complex type definition. The consequence of this

declaration is that an element called comment may appear in an instance document, and its content must be consistent with that element's type.

(5) Occurrence Constraints

The state element is **optional** within recordType (see Figure 2.11.) because the value of the minOccurs attribute in its declaration is **0**. In general, an element is **required** to appear when the value of minOccurs is **1** or **more**. The **maximum number** of times an element may appear is determined by the value of a **maxOccurs** attribute in its declaration. This value may be a **positive integer** such as **41**, or the term **unbounded** to indicate there is **no maximum number of occurrences**. The **default value** for both the minOccurs and the maxOccurs attributes is **1**. Thus, when an element such as comment is declared without a maxOccurs attribute, the element may not occur more than once. Be sure that a specify value for only the minOccurs attribute is less than or equal to the default value of maxOccurs, i.e. it is 0 or 1. Similarly, a specify value for only the maxOccurs attribute must be greater than or equal to the default value of minOccurs, i.e. 1 or more. If both attributes are omitted, the element must appear exactly once.

Attributes may appear **once** or **not at all**, but **no other number of times**, and so the **syntax** for **specifying occurrences** of attributes is **different** than the syntax for elements. In particular, attributes can be declared with a **use** attribute to indicate whether the attribute is required. For example, phone attribute declaration in Figure 2.11. is **optional** or even **prohibited**.

Default values of both attributes and elements are declared using the

default attribute, although this attribute has a slightly different consequence in each case. When an attribute is declared with a default value, the value of the attribute is whatever value appears as the attribute's value in an instance document; if the attribute does not appear in the instance document, the schema processor provides the attribute with a value equal to that of the default attribute. Note that default values for attributes only make sense if the attributes themselves are optional, and so it is an error to specify both a default value and anything other than a value of optional for use.

(6) Global Elements and Global Attributes

Global elements, and **global attributes**, are created by **declarations** that appear as the **children** of the **schema element**. Once declared, a global element or a global attribute **can be referenced** in one or more declarations using the **ref attribute** as described above. A declaration that references a global element enables the referenced element to appear in the instance document in the context of the referencing declaration. So, for example, the comment element at the same level as the mortgagee, mortgages and bank elements because the declaration that references comment appears in the complex type definition at the same level as the declarations of the other three elements.

One caveat is that **global declarations** cannot contain **references**; global declarations must **identify simple** and **complex types** directly. Put concretely, **global declarations** cannot contain the **ref attribute**, they must use the **type attribute**. A second caveat is that **cardinality constraints** cannot be placed on global declarations, although they can be placed on

local declarations that reference global declarations. In other words, global declarations cannot contain the attributes **minOccurs**, **maxOccurs**, or **use**.

(7) Specifying Uniqueness

XML Schema enables developer to indicate that any **attribute** or **element value** must be **unique** within a **certain scope**. To indicate that one particular attribute or element value is unique, developer use the **unique** **element** first to **"select"** a set of elements, and then to **identify** the **attribute** or **element "field"** relative to each selected element that has to be unique within the scope of the set of selected elements. The **selector element's xpath attribute** contains an **XPath expression** that selects a list of all the elements in an instance. Likewise, the field **element's xpath attribute** contains a second **XPath expression** that specifies that the **attribute values** of those elements must be **unique**. Note that the XPath expressions limit the scope of what must be unique.

Developer can also indicate combinations of fields that must be unique, by specifying that for each item element. The combined values of its partNum attribute and its productName child must be unique. To define combinations of values, simply use multiple field elements to identify all the values.

(8) Defining Keys and Keys' Reference

To define keys and keys' reference researcher uses MortgageNew.xsd (see Figure B.1.) for the completed schema, because Mortgage.xsd is hierarchical XML so it does not define key and key reference. Developer could enforce the constraint using unique, however, developer also want to ensure that every mortgages element listed under a **documentDate** has a

corresponding document description. The enforce constraint use the key and keyref elements. The MortgageNew.xsd (see Figure 2.13.) shows that the **key** and **keyref** constructions are applied using almost the same syntax as **unique**. The key element applies to the **documentDate** attribute value of document elements that are children of the document element. This declaration of **documentDate** as a key means that its value must be unique and cannot be set to nil (i.e. is not nillable), and the name that is associated with the key, **documentKey**, makes the key referenceable from elsewhere.

To ensure that the **document_mortgage** elements have corresponding document descriptions, developer says that the **documentDate** attribute (`<field xpath="documentDate"/>`) of those elements (`<selector xpath="//mortgage"/>`) must reference the **documentKey** key. The **documentDate** declaration as a keyref does not mean that its value must be **unique**, but it means there must **exist** a **documentKey** with the same value.

```

52 <xsd:key name="documentKey">
53   <xsd:selector xpath="//document" />
54   <xsd:field xpath="documentDate" />
55 </xsd:key>
. . .
64 <xsd:key name="mortgageKey">
65   <xsd:selector xpath="//mortgage" />
66   <xsd:field xpath="loanNumber" />
67 </xsd:key>
. . .
76 <xsd:keyref name="document_mortgage" refer="documentKey">
77   <xsd:selector xpath="//mortgage" />
78   <xsd:field xpath="documentDate" />
79 </xsd:keyref>
. . .

```

Figure 2.13. The Listing of Keys and Keys' Reference Definition in MortgageNew XML Schema.

(9) XML Editor and XML Checkers

XML Schemas and XML Documents can be created by using any XML editor (Holzner 2004) (Wyke 2002), such as XML Pro (<http://www.vervet.com>), Microsoft's Visual Studio .NET, Turbo XML (http://www.extensibility.com/downloads/trial_downloads.htm), XML Spy (<http://www.xmlspy.com>), XML Writer (<http://xmlwriter.net>), and XML Notepad (<http://www.webattack.com/get/xmlnotepad.shtml>). Several XML editors give a feature to validate an instance document against a schema. They can spot basic XML syntax error and can indicate that the syntax is well-formed, but are incapable of providing information about the correctness or incorrectness of a created schema. An online XSD schema checking service is available using XML schema validator provided at the W3C Web site. The schema validation service for the May 2001 Recommendation is located at www.w3.org/2001/03/webdata/xsv. Essentially, the schema needs to be accessible via a URL in order to be validated.

Researcher will use XML Writer and Microsoft's Visual Studio .NET in study because each has the different capability. For checking well-formed and validated researcher prefer XML Writer. For instance, Figure C.1. shows the checked well-formed MortgageNew XML schema output and Figure C.3. shows the checked validated MortgageNew Xml document output. However, for visualization XML schema and XML document the researcher desire Microsoft's Visual Studio .NET. For example, see Figure 2.14., the visualization of XML Schema in Figure A.1.

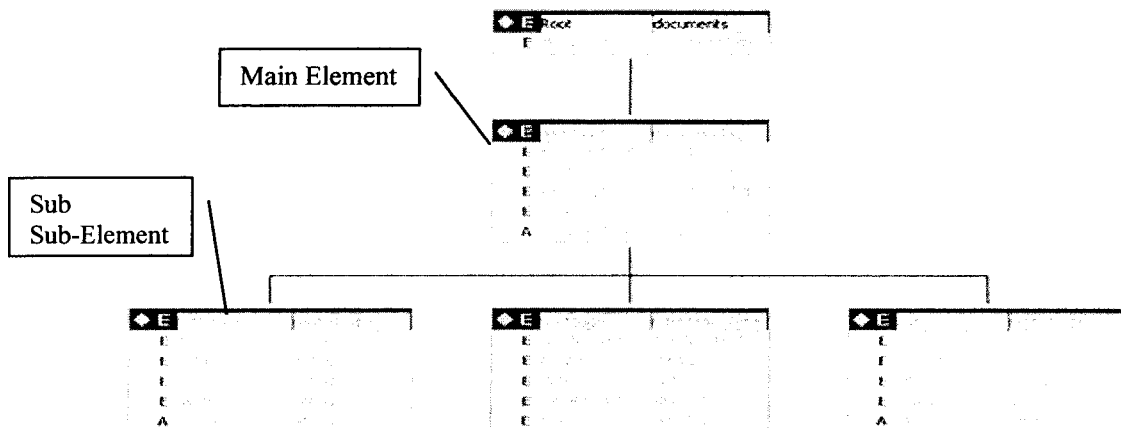


Figure 2.14. A Visualization Hierarchical Schema for Mortgage XML Schema.

However, beside XML Editors that mentioned above, several database application over a feature to transform database to XML and vice versa. In general database type products are relational, however, a view as hierarchical or ISAM. In this occasion, researcher shows product categories middleware (see Table 2.3.) and XML-enabled databases (see Table 2.4.). The complete product categories can be shown in <http://www/rpbouret.com/xml/XMLDatabaseProds.htm>. As a note, middleware means that software can be called from application to transfer data between XML documents and databases. Likewise, XML-enabled database is the database with extensions for transferring data between XML documents and themselves.

(10) Simple Types

Some of these simple types, such as string and decimal (the complete simple type built in XML Schema and example to define simple type see <http://www.w3.org/TR/2004/PER-xmlschema-0-20040318>, page 20 until page 29), are built in to XML Schema, while others are derived from the

built-in's. New simple types are defined by deriving them from existing simple types (built-in's and derived). In particular, developer can derive a new simple type by restricting an existing simple type, in other words, the legal range of values for the new type are a subset of the existing type's range of values. Developer uses the **simpleType element to define and name the new simple type**. Developer uses the **restriction element to indicate the existing (base) type**, and to **identify the "facets"** that constraint **the range of values**. A complete list of facets is provided in Table 2.2.

Table 2.3. Product Middleware.

Product	Developer	License	DB Type	DB→XML	XML→DB
ADO	Microsoft	Commercial	Relational	X	X
Attunity Connect	Attunity Ltd.	Commercial	Relational, hierarchical	X	X
Data Junction	Data Junction, Inc.	Commercial	Relational, ISAM	X	X
iWay XML Transformation Engine (iXTE)	iWay Software	Commercial	Relational, hierarchical	X	X
XML Gateway	SPI Ltd.	Commercial	Relational, Excel, Word, text files	X	X

Table 2.4. XML-Enabled Databases.

Product	Developer	License	DB Type
Access 2002	Microsoft	Commercial	Relational
Informix	IBM	Commercial	Relational
Oracle 8i and 9i	Oracle	Commercial	Relational
SQL Server 2000	Microsoft	Commercial	Relational
Matisse	Matisse Software	Commercial	Object-oriented

Suppose a developer wishes to create a new type of integer called `myInteger` whose range of values is between 10000 and 99999 (inclusive). The developer bases our definition on the built-in simple type `integer`, whose range of values also includes integers less than 10000 and greater than 99999. To define `myInteger`, the developer restricts the range of the integer base type by employing two facets called `minInclusive` and `maxInclusive` (see Figure 2.15.).

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 2.15. The Listing of the `myInteger` Range 10000-99999 Definition.

The example shows one particular combination of a base type and two facets used to define `myInteger`, but a look at the list of built-in simple

types and their facets should suggest other viable combinations.

The loanNumber schema contains another, more elaborate, example of a simple type definition. A new simple type called loanNumberType is derived (by restriction) from the simple type string. Furthermore, developer constraint the values of loanNumber using a facet called pattern in conjunction with the regular expression "\d{2} \d{4} \d{2}" that is read "two digits followed by a space followed by four digits followed by a space finally followed by two digits " (see Figure 2.16.).

```
<xsd:simpleType name="loanNumberType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{2} \d{4} \d{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 2.16. The Listing of the Simple Type "loanNumberType" Definition.

XML Schema defines twelve facets which are listed in Table 2.2. Among these, the enumeration facet is particularly useful and it can be used to constraint the values of almost every simple type, except the boolean type. The enumeration facet limits a simple type to a set of distinct values. For example, developer can use the enumeration facet to define a new simple type called USState, derived from string, whose value must be one of the standard US state abbreviations (see Figure 2.17.).

USState would be a good replacement for the string type currently used in the state element declaration. By making this replacement, the legal values of a state element, i.e. the state sub-elements of state, would be

limited to one of AK, AL, AR, etc. Note that the enumeration values specified for a particular type must be unique.

```
<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
    <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>
```

Figure 2.17. The Listing of the Enumeration Facet Usage.

There are two possibilities to represent data structure in XML (Daum 2003), i.e. Hierarchical XML or Flat XML. In **Hierarchical XML**, the **relations** between the **data elements** are represented through the **implicit hierarchical relations** between **parent** and **child** elements. For instance, see Figure 2.14. The figure visualizes Hierarchical Mortgage XML Schema in Figure A.1. The structure allows duplication child element data in XML Document in order to support HTML for presentation data easily.

Ramakrishnan (2003) in his book discusses mapping XML Schemas to Relation Schemas. Implicitly he shows how to mapping Hierarchical XML Schemas to Relation Schemas using **hierarchical sequence**, as mentioned by Elmasri (see section 2.1. (4)). The mapping for Hierarchical Mortgage XML Schemas in Figure 2.14. are DOCUMENT (documentDate, Comment, **document_Id**), MORTGAGEE (name, location, city, state, phone, **document_Id**), BANK (name, location, city, state, phone, **document_Id**), and MORTGAGE (property, date, loanAmout, term, loanNumber, **document_Id**).

Flat XML, however, the relations between the data elements are represented through the relations **explicitly** via **key** and **keyref elements**. The structure of the document is established via matching key and keyref, for instance (see Figure 2.18.). The figure visualizes Flat MortgageNew XML Schema in Figure B.1. This is done in a way similar to relational database. The data is normalized and finally represented as an interrelated network of “flat” elements. Flat XML is suitable for data transfer Daum (2003) and Bourret (2004). The advantage of the Flat XML is that can use the well-known relational techniques to keep the design of such a documents sound. The disadvantage is that the XML Document is hard to read. Also it is not shorter than the original document-on the contrary, it is longer. Constructing such a XML Schema also requires some bookkeeping (to allocate unique keys to elements), and retrieving information from such a document requires much cross-referencing and joining.

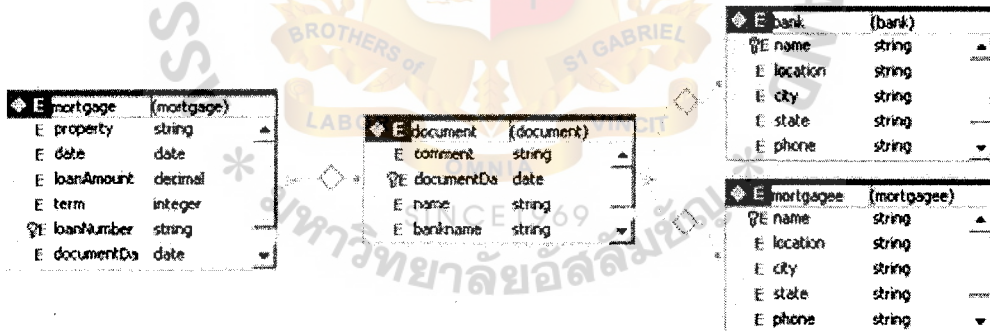


Figure 2.18. A Visualization Flat Schema for MortgageNew XML Schema.

Researcher wants to show the differences in defining Hierarchical XML Schema and Flat XML Schema. For example, the comparison of Hierarchical Mortgage XML Schema and Flat MortgageNew XML Schema is shown in Figure 2.19. The key point is line 2 in Figure 2.19. (a). To become the Flat one see line 2 and 3 in Figure 2.19. (b).

Furthermore, complex type definition on line 14 until line 22 in Figure 2.19. (a) become complex type definition on line 4 until line 58 in Figure 2.19. (b) with additional key point line **<xsd:choice maxOccurs="unbounded">** and **</xsd:choice>** respectively in line 5 and 57. In addition, generally Hierarchical XML is used to exchange one instance main-element and several sub-elements. Therefore, many text books do not discuss how to exchange or transfer several main-elements instance (forest). Since still having several product type hierarchical (see Table 2.2.), the researcher tries to give a proposed solution (see Figure 2.19. (a) line 8 until line 13 and Figure A.2. for full listing of Hierarchical Mortgage XML Document). Without line 8 until line 13 only one instance can be saved in the XML Document, the researcher uses XML Writer as an editor and a validator in experiment.

2.3 Nijssen's Information Analysis Methodology (NIAM)

There are several reasons the researcher choose the NIAM. First, the NIAM is a well established conceptual schema model. Second, the NIAM pays attention to details, such as functional dependencies between attributes because the NIAM is a fact-oriented conceptual schema model (see section 2.3 (2)). Finally, the NIAM has a transformation algorithm to transform the NIAM conceptual schema into fifth normal form relational schemas (see section 2.3 (3)).

In addition, Halpin (1995) mentions that the NIAM can simplify the design process by using natural language, intuitive diagrams and examples, and by examining the information in terms of simple or elementary facts. By expressing the model in terms of natural concepts, like objects and roles, it provides a conceptual approach for modeling. The NIAM has been originally developed by Professors G. M. Nijssen and E. D. Falkenberg. Latter T.A. Halpin has revised and extended the NIAM.

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3      .
4      .
5      .
8      <xsd:element name="Root" type="documents" />
9      <xsd:complexType name="documents">
10         <xsd:sequence>
11             <xsd:element name="document" type="documentType"
minOccurs="0" maxOccurs="unbounded" />
12         </xsd:sequence>
13     </xsd:complexType>
14     <xsd:complexType name="documentType">
15         <xsd:sequence>
16             <xsd:element ref="comment" minOccurs="1" />
17             <xsd:element name="mortgagee" type="recordType" />
18             <xsd:element name="mortgage" type="mortgageType"
minOccurs="0" maxOccurs="8"/>
19             <xsd:element name="bank" type="recordType" />
20         </xsd:sequence>
21         <xsd:attribute name="documentDate" type="xsd:date" />
22     </xsd:complexType>
23     <xsd:complexType name="recordType">
24         .
25         .
26         .
31     </xsd:complexType>
32     <xsd:complexType name="mortgageType">
33         .
34         .
35         .
52 </xsd:complexType>
59 </xsd:schema>

```

To become
a Forest
document

(a) Listing Hierarchical Mortgage XML Schema

Figure 2.19. The Listing of a Comparison between the Hierarchical XML Schema and the Flat XML Schema.

An information system for a given application may be looked at three levels, i.e.: conceptual, external, and internal. At each level the formal model or knowledge base comprises a schema which describes the structure or design of the universe of discourse (UoD), and a database which is populated with the fact instances. Each schema specifies what states and transitions are permitted for its database. The conceptual schema does this in terms of simple, human-oriented concepts.

A conceptual schema comprises three main sections, i.e.: stored fact types, constraints, and derivation rules. Fact types are indicated by what kinds of the object

there are, how these are referred and what roles it plays. Each role in a relationship is played by only one object type. The simplest kind of object is a value. Entities are real or abstract objects which are identified by their relationship to values. Constraints restrict the populations and transitions of the fact types. Derivation rules enable further facts to be derived from the facts that are stored.

```

1 <?xml version="1.0" ?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
3   <xsd:element name="NIAM" msdata:IsDataSet="true" >
4     <xsd:complexType>
5       <xsd:choice maxOccurs="unbounded">
6         <xsd:element name="document">
7           . . .
15          </xsd:element>
16          <xsd:element name="mortgagee">
17            . . .
26            </xsd:element>
27            <xsd:element name="bank">
28              . . .
37              </xsd:element>
38              <xsd:element name="mortgage">
39                . . .
49                </xsd:element>
50              </xsd:choice>
51            </xsd:complexType>
52            <xsd:key name="documentKey">
53              <xsd:selector xpath="."//document" />
54              <xsd:field xpath="documentDate" />
55            </xsd:key>
56            <xsd:key name="mortgageeKey">
57              <xsd:selector xpath="."//mortgagee" />
58              <xsd:field xpath="name" />
59            </xsd:key>
60            <xsd:keyref name="document_mortgagee" refer="mortgageeKey">
61              <xsd:selector xpath="."//document" />
62              <xsd:field xpath="name" />
63            </xsd:keyref>
64          . . .
80        </xsd:element>
81      </xsd:schema>

```

To change Complex Type document in Listing 2.9 (a)

(b) Listing Flat MortgageNew XML Schema

Figure 2.19. The Listing of a Comparison between the Hierarchical XML Schema and the Flat XML Schema (Continued).


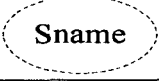
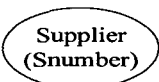

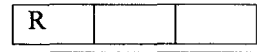

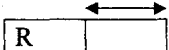
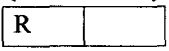

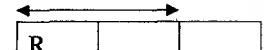

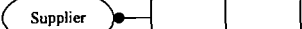
Main concepts in the NIAM include entity types, label types (or value types), fact types, and reference types (or bridge type) (see Table 2.5.). Similar to other semantic data model, the entity in the NIAM is an object of interest. Entities which have the same properties form entity types. However, there are no concepts of attribute. Instead, there are label types which are types of label or values associated with each entity type.

A relationship between an entity type and a label type is called a reference type. Each entity type may have more than one reference type associated with it but there must be at least one one-to-one reference type for identification purpose. A selected one-to-one reference type is called the unique identifier. Figure 2.20. shows the entity type Supplier with its unique identifier Snumber and a many-to-one reference type with the label type Sname. A fact type is a relationship between entity types. Fact type must be elementary (can not be further decomposed) since each of it is based on a deep structured natural language sentence. Fact types and reference types can be collectively called relationship type.

(1) Unique Identifier

A unique identifier is a minimal combination of columns where no duplicates are allowed. The guideline for select the unique identifier when two or more candidate identification schemes exist: first, minimize the number of labels needed. Secondly, favor an identifier which is more stable. The third criterion for selecting an identifier is easy for user to recognize. Selecting a primary identifier from two or more candidate identifiers might be regarded as an implementation decision rather than as a conceptual issue.

Table 2.5. Some Basic Symbols Used in Conceptual Schema Diagram.

Symbol	Explanation
	Entity type Supplier
	Label types or value type Sname
	Supplier identified by means of 1:1 reference type Snumber
	Binary predicate R (2 roles)
	Ternary predicate R (3 roles), etc.
	Uniqueness constraint on left role of R, N:1 association
	Uniqueness constraint on right role of R, 1:N association
	Uniqueness constraint on combination of roles, N:M association
	Uniqueness constraint on each role, 1:1 association
	Uniqueness constraint on role pair 1-3
	Role optional for population of Supplier
	Role mandatory for population of Supplier

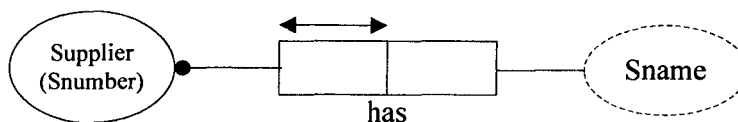


Figure 2.20. A Reference type in NIAM.

(2) Conceptual Schema Design Procedure (CSDP) Methodology

CSDP can be employed to describe the UoD in terms of a conceptual schema. There are seven steps in the CSDP, i.e.:

- (A) Transform familiar information examples into elementary facts, and apply quality checks.
- (B) Draw the fact types, and apply a population check.
- (C) Check for entity types that should be combined, and note any arithmetic derivations.
- (D) Add uniqueness constraints, and check arity of fact types.
- (E) Add mandatory role constraints, and check for logical derivations.
- (F) Add value, set comparison and sub typing constraints.
- (G) Add other constraints and perform final checks.

In this section, the researcher applies CSDP to the Suppliers and Parts Database example in Figure 2.1., to show how the procedure usually works. The first steps: transform information in Figure 2.1. into elementary facts, such as: the supplier with supplier number “S1” has a name “Smith”, the supplier with supplier number “S1” stay in a city name “London”, the city with name “London” has a status number “20”, the part with part number “P1” has a name “Nut”, the part with part number “P1” is stored in a city with name “London”, the part with part number “P1” has a color “Red”, and the stock for the supplier with supplier number “S1” and the part with part number “P1” is “300”.

The second step: draw a conceptual schema diagram which shows all the fact types mentioned above (see Figure 2.21.). The diagram illustrates the relevant object types, predicates, and references schemas. Next task in

this step applies a population check. As a checking the diagram is correct, each fact type on the diagram must be populated with the original fact instances. To do this is, by adding a fact table for each fact type and entering the values in the relevant columns of this table. The resulting diagram is called a **knowledge base diagram**, since it shows both schema and a sample database (see Figure 2.21.).

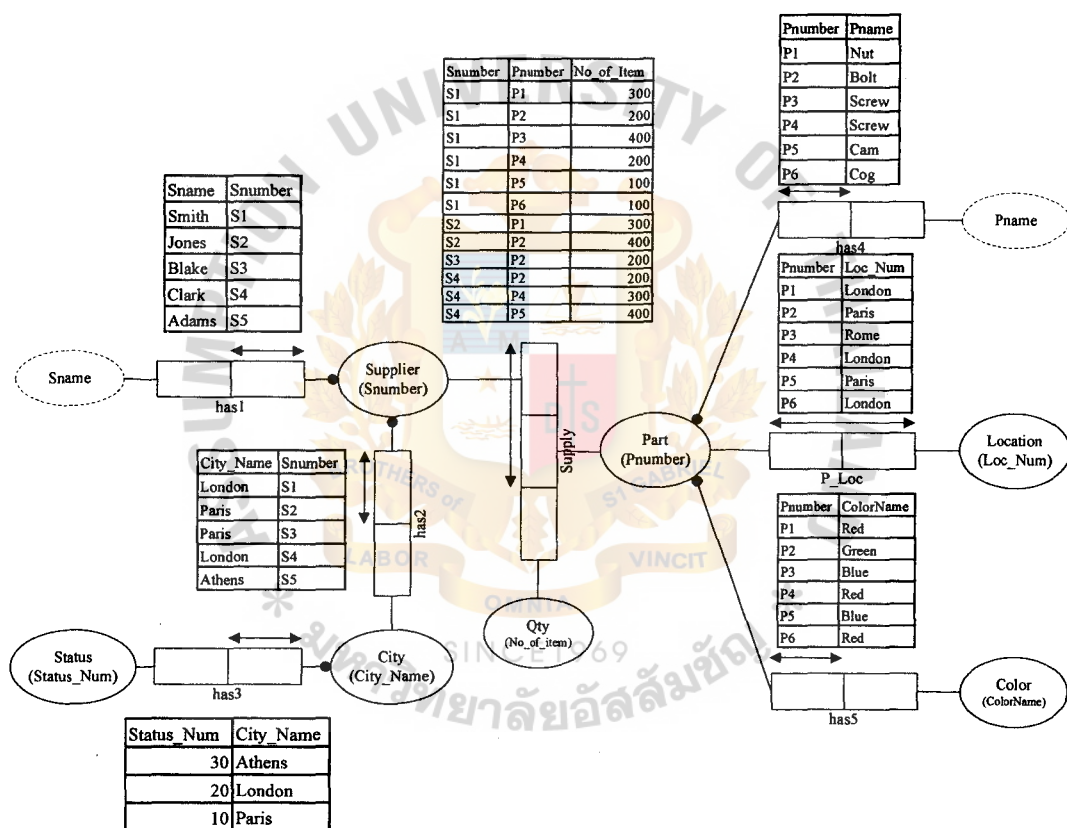


Figure 2.21. Suppliers and Parts Knowledge Base Diagram.

The third step: check for entity types which should be combined, and check fact types can be derived from the others by arithmetic computation. For example, the entity type supplier name and the entity type product name can be combined. Since no fact types can be derived from arithmetic

computation, it can be shown.

The fourth steps: add uniqueness constraints, and check arity of fact types. See Figure 2.21, the uniqueness constraints appears as an **arrow-tipped bar** on the upper of fact type. The result of this step plays a pivotal role when the conceptual schema is later mapped onto a relational schema. Once uniqueness constraints have been added to a fact type, some further checks are made to see whether the fact type is of the right arity or length. In particular, there is a simple check based on uniqueness which shows that certain kinds of fact types are not elementary and hence should be split. The role will be adopted for reverse engineering on the propose software tools.

The fifth step: add mandatory role constraints and check for logical derivation. Basically, these indicate which roles must be played by the population of an object type, and which are optional. Once mandatory role constraints have been specified, a check is made to see if some fact types may be logically derived from others. A role is **mandatory** if and only if, for all states of the knowledge base, the role must be played by every member of the population of the attached object type; otherwise the role is **optional**. See Figure 2.21., **dot** show as a mandatory.

The sixth step: add value, set comparison and sub typing constraints. In this step, the value constraints (i.e. restriction on value types) are defined. For example, in Suppliers and Parts database about color products the values of Color might be restricted to “R”, “G”, and “B” (for Red, Green, and Blue). This step also specifies a format pattern, for instance: <c20> for supplier name, it means that a sting of a most 20 characters. The last step CSDP, add other constraints and performs final checks. The final

check to ensure the information model is consistent and free from redundancy.

The developer might declare fact type too long or too short in the first step CSDP. The problem will be fixed in the fourth step. Too long means that the arity of fact type is higher than it should be - the predicate has too many roles. In this case the fact type must be split into two or more simple fact types. Too short, means that the arity of some fact types is too small. In this case, the relevant fact types are combined into one of higher arity. As a guideline see Table 2.6., **an n-ary fact type has a key length of at least n-1**. Halpin says if this rule is violated, the fact type is not elementary, and hence should be split by using functional dependency concept.

By pessimistic, all columns become **one fact type**. The researcher shows how **CSDP is adopted** in the process **reverse engineering**. The CSDP does not start from the first step as usual, however, the CSDP will be start from the **fourth step**, add uniqueness constraints and check arity of fact types, with assumption XML Document is significant. For example, see Figure 2.22. The figure shows quaternary fact type with one uniqueness constraint, Snumber. See Table 2.6., fact type with 4 arity must have minimum key length 3. It means too long, therefore the fact type must be split using functional dependency concept, such as Snumber \rightarrow Sname, Snumber \rightarrow City_Name, and Snumber \rightarrow Status_Num. Basically, every produced functional dependency is the fact type that must be created in the CSDP first step. For example, Snumber \rightarrow Sname the fact type means **the supplier with supplier number "S1" has a name "Smith"**. After that the

step will be followed by the CSDP second, third, fifth, sixth, and seventh step.

Table 2.6. Key Length Check for Ternaries and Longer Fact Types.

Arity of fact type	Minimum key length	Illegal key lengths
3	2	1
4	3	1, 2
5	4	1, 2, 3
...		
N	N-1	1, ... , N-2

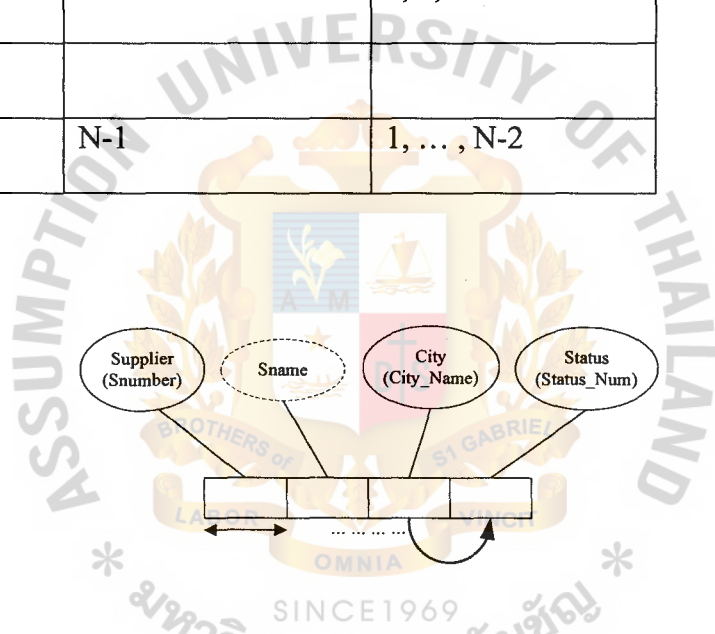




Figure 2.22. A Quaternary Fact Type That is Splitable (FD Added).

(3) Relational Mapping (Rmap) Procedure

With the Rmap procedure, Halpin (1995) guarantees a redundancy-free relational design, and includes strategies to restrict the number of tables. The Rmap extends and refines an older mapping procedure known as the optimal normal form algorithm. Although the Rmap procedure is not necessarily but the last word in table design is extremely valuable since it guarantees a safe and reasonably efficient design. Recall that redundancy is

repetition of an elementary fact. Having gone into the trouble of ensuring that conceptual fact types are elementary, it can very easily avoid redundancy in relational tables.

Since each relational table stores one or more elementary fact types, it can automatically avoid redundancy by ensuring that each fact type maps into only one table, in such a way that its instances appear only once. To achieve this, there are two rules, as follows:

- (A) Fact types with compound uniqueness constraints  map to separate tables.
- (B) Fact types with functional roles attached to the same object type  are grouped into the same table, keyed on the object type's identifier.

Group of fact types with functional roles attached to the same object type or group of fact types with the same compound uniqueness constraints, in the study is called **main entity type**. The two rules above show how to map main entity type into table schemes. Any predicate, other than an objectified predicate, which has a uniqueness constraint spanning two or more of its roles must map to a table by itself. Hence many-to-many binaries, and all n-aries ($n \geq 3$) on the conceptual schema, map to a separate table.

A fourth option for one-to-one case uses two tables, but include the one-to-one binary in both, with a special equality constraint to control the redundancy. A default procedure is summarized in Figure 2.23. Each one-to-one fact type maps to only one table:

if only one object type in the one-to-one predicate has another

functional role

then group on its side {case (a)}

else if both object types have other functional roles and only one role in the one-to-one is explicitly mandatory

then group on its side {case (b)}

else if no object type has another function role

then map the one-to-one to a separate table

else grouping choice is up to developer

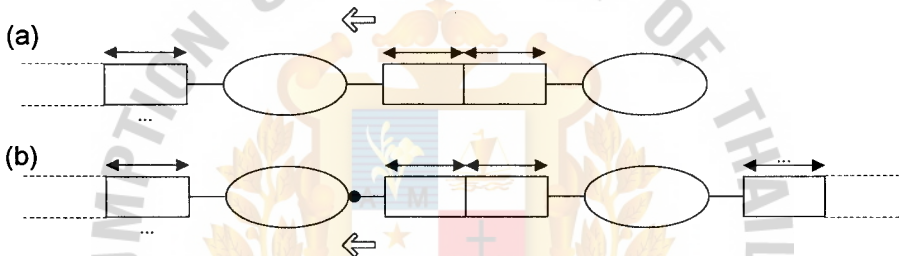


Figure 2.23. Default Procedure for Mapping One-to-One Fact Types.

The researcher applies the roles to Suppliers and Parts schema diagram in Figure 2.21., as a result see Figure 2.24. To make clear understanding, the researcher has separated the schema diagram into category role, single uniqueness constraints and compound uniqueness constraints. The left side figure applies the first role and the right side figure applies the second role. Convention in the study, the underlined is the primary key and the italic is the foreign key.

There is no one-to-one in Suppliers and Parts database, however, the researcher just wants to explain how to implement. The researcher shows in

fact type between entity Supplier and City. In case, one supplier can exist in only one City and also one City only has one Supplier (see Figure 2.25.). In brief, the one-to-one relationship mandatory is very important in mapping.

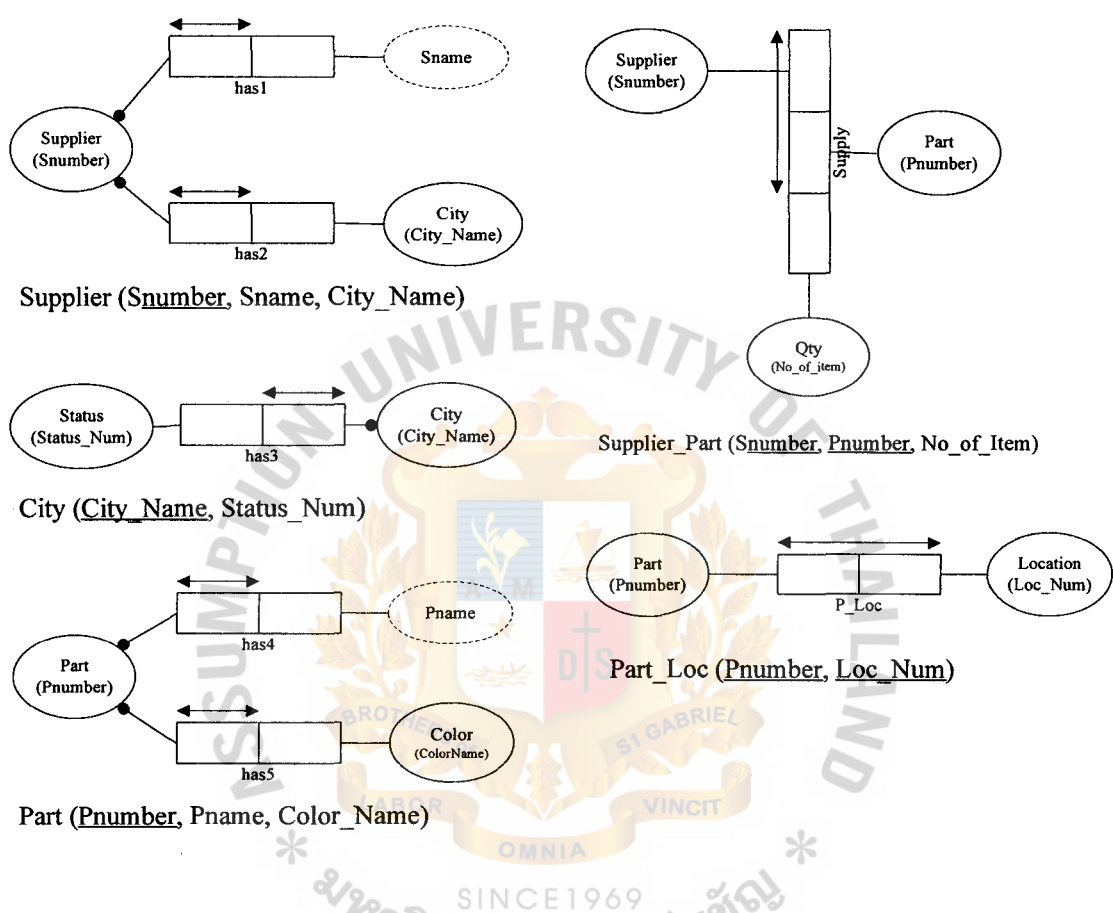


Figure 2.24. Suppliers and Parts Relational Mapping from NIAM Conceptual Schema into Relation Schema.

2.4 Visual Basic (VB).NET and ActiveX Data Objects (ADO).NET

There are several reasons the researcher chooses VB.NET and ADO.NET to support the study. First, VB.NET is an easier-to-use integrated development environment. Second, VB.NET supports Web Forms and XML Web Services. Therefore, in VB.NET, the researcher can use ADO.NET to read and write XML simply. Lastly, VB.NET has a feature Windows Forms that provide a clear, object-

oriented, extensible set of classes enabling the researcher to develop rich Window applications. In addition, with Microsoft's Visual Studio .NET, the researcher can edit, validate, and visualize XML Schema and XML Document.

XML is one of the hot technologies in the development world right now. At present, developers need to build more powerful applications. For example, more and more developers want to work with XML data. ADO.NET is designed to support XML, to disconnect data access easily, to more control over updates, and to greater update flexibility. One major goal of the ADO.NET development team is to bridge the gap between XML and data access so that developer can easily integrate the two technologies. Loading data from an XML document into an ADO.NET DataSet and vice-versa is simply.

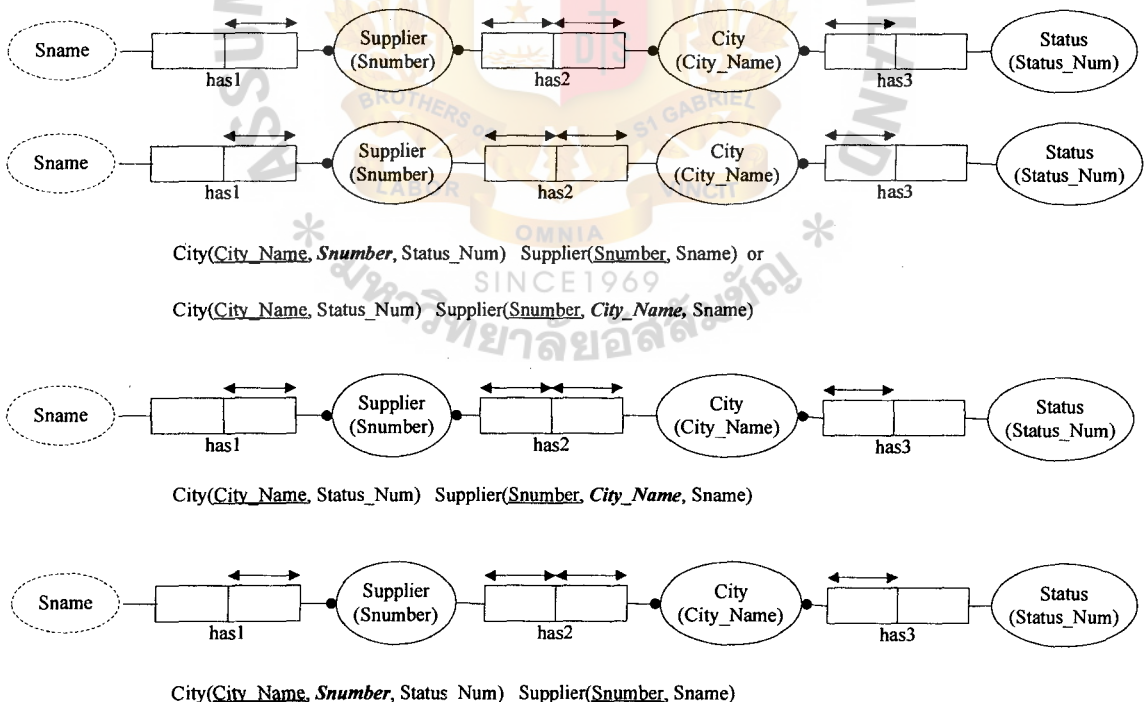


Figure 2.25. Supplier and City Relational Mapping for One-to-One Relationship.

ADO.NET is a set of libraries included with the Microsoft .NET framework that helps developers to communicate with various data stores from .NET applications (Sceppa 2002). The ADO.NET objects are divided into Connected Objects and Disconnected Objects. The Connected Objects libraries include classes for connecting to a data source, submitting queries, and processing results (see Figure 2.26.). Moreover, it can also use as a robust, hierarchical, disconnected data cache to work with data off line. The creating software tools use disconnected ADO.NET because the researcher assumes that the transferred XML Schemas and XML Documents have already created disconnect to the data source.

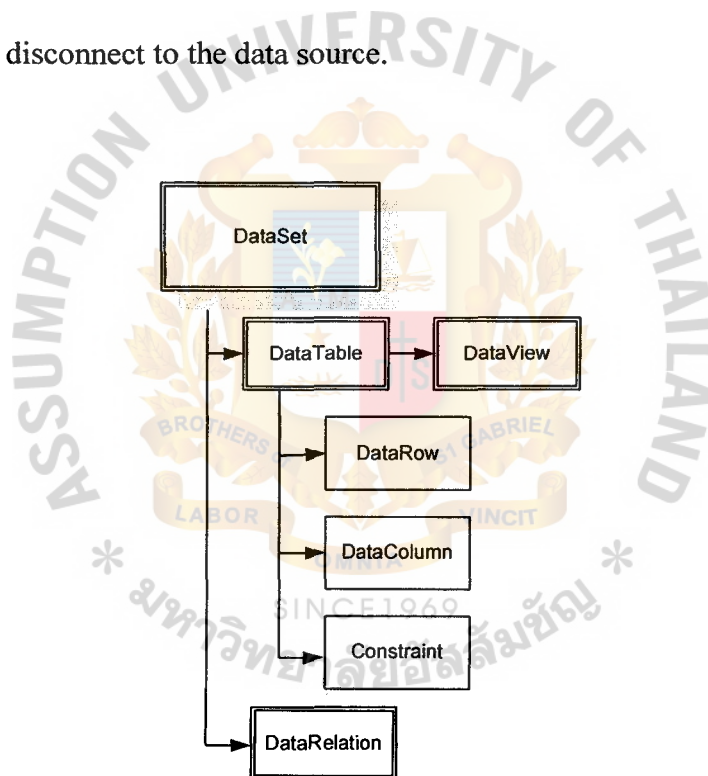


Figure 2.26. The Disconnected ADO.NET Objects Hierarchy.

(1) DataSet

The central disconnected-object ADO.NET is **DataSet** object. As its name indicates, it contains a set of data. It has several features that allow developers to write it to and read it from a file or an area of memory. In

addition, DataSet allows the developer to sort, search, filter, store a pending change, and navigate through hierarchical data. Furthermore, the DataSet also includes a number of features that bridge the gap between traditional data access and XML development. Developers can now work with XML data through traditional data access interfaces and vice-versa.

The ADO.NET DataSet has been built from the ground up to work with XML. Developer can save and load the contents of a DataSet to and from files as XML documents. In discussion, DataSet will be called by **DataSet name**, for instance **DocumentDS DataSet** will be called by **DocumentDS**. DS is the abbreviation DataSet. The DataSet also lets developers separate the schema information (table, column, and constraints information) into an XML schema file.

In ADO.NET, DataSet objects and XML documents are almost interchangeable. It is easy to move from one data structure to the other. This duality allows developers to use the interfaces most comfortable. XML programmers can work with DataSet objects as XML documents, and database programmers can work with XML documents as Dataset objects.

The DataSet object has a series of methods that let developer examine the contents of XML, as well as load XML data into the DataSet. The DataSet object's **WriteXml method** is used to **write** the contents of **DataSet** to a **file** or to an **object** that implements the **Stream, TextWriter, or XmlWriter interfaces**. The dataset object has a **ReadXml method** as well to **load data** into DataSet. It can read XML data from a **file** or from an **object** that implements the **Stream, TextReader, or XmlReader interfaces**.

The **dataSet** object also exposes **ReadXmlSchema** and **WriteXmlSchema** methods that allow developers to **read** and **write** the schema information for DataSet. Each method supports working with **files** and **objects** that implement the **Stream**, **TextReader**, or **XmlReader** interface.

(2) DataTable

The ADO.NET **DataTable** object allows developers to examine data through **collection** of **rows** and **columns**. For instance, the contents of a **DataTable** can be accessed through its **Rows** property, which returns a collection of **DataRow** objects. The structure of a DataTable can be examined through **Columns** property. The DataTable class also lets to define constraints, such as **primary key**.

Each DataTable has a collection, which is a container for **DataColumn** objects. As its name implies, a DataColumn object corresponds to a **column** in a **table**. It stores information about the **structure of the column**. This type of information, data about data, is called **metadata**. For example, a DataColumn exposes a **type** property that describes the **data type** (such as string or integer) that the column stores. The DataColumn has other properties such as **readOnly**, **AllowDBNull**, **Unique**, **Default**, and **AutoIncrement** that allow developers to control whether the data in the column can be updated, restrict what can be stored in the column, or dictate how values should be generate for new rows of data. The other properties see Table 2.7.

Table 2.7. Properties of the DataColumnn Object.

Property	Data Type	Description
AllowDBNull	Boolean	Controls whether the column will accept null values
AutoIncrement	Boolean	Controls whether ADO.NET will generate new autoincrement values for the column
AutoIncrementSeed	Integer	Controls what value ADO.NET use for the first new autoincrement value
AutoIncrementStep	Integer	Controls the value ADO.NET will use to generate subsequent autoincrement values
Caption	String	Controls the caption of the column when displayed in a bound data grid
ColumnMapping	MappingType	Controls how ADO.NET will store the contents of the column in an XML document
Expression	String	Controls how ADO.NET will generate values for expression-based columns
DataType	Type	Controls the data type that ADO.NET will use to store the contents of the column
ColumnName	String	Contains the name of the DataColumnn object
DefaultValue	Object	Controls the default value that ADO.NET will use to populate this column for new rows

Table 2.7. Properties of the DataColumn Object (Continued).

Property	Data Type	Description
ExtendedProperties	PropertyCollection	Contains a collection of dynamic properties and values
MaxLength	Integer	Specifies the maximum length of the string that the column can contain
Namespace	String	Contains the namespace that ADO.NET will use when it writes the content of the DataSet to XML when it loads XML data into DataSet
Ordinal	Integer	Return the index of the DataColumn within the DataTable object's columns collection
Prefix	String	Contains the prefix for the namespace that ADO.NET will use when it writes the contents of the DataSet to XML when it loads XML data into DataSet
ReadOnly	Boolean	Controls whether the contents of the column are read-only
Table	DataTable	Returns the DataTable to which the DataColumn belongs
Unique	Boolean	Controls whether ADO.NET requires that the values for the column be unique within the DataTable

The DataTable class also provides a way to place constraints on the data. For example, developers can build a **Constraint object** that **ensures** the **values** in a **column** or **multiple columns** are **unique** within the DataTable. Constraint objects are maintained in a DataTable object's Constraints collection.

To access the **actual value** stored in a DataTable object, developer use the object's **Rows collection**, which contains a series of DataRow objects. To examine the data stored in a specific column of a particular row, developers use the Item property of the appropriate DataRow object to read the value for any column in that row. The DataRow class provides several overloaded definitions of its Item property. Developers can specify which column to view by passing the column name, index value, or associated DataColumn object to a DataRow object's Item property. Item is the default property of the DataRow object so developers can use it implicitly.

Related with XML Schemas which will be reengineering with the NIAM conceptual schema, every **complex type** in XML Schemas becomes a **DataTable** in ADO.NET and every **element** in complex type becomes a **DataColumn** of DataTable in ADO.NET. Therefore, the term **main element** in XML Schemas becomes **parent table** in ADO.NET and equally for **sub element** becomes **child table** (see section 2.2 (1) and (2)) for **Hierarchical XML**. However, for **Flat XML** every **complex type** becomes a **DataTable** and every **element** become a **DataColumn**. Furthermore, related with XML Documents that validate with the XML Schemas, all elements instant in a complex type become a DataRow of DataTable in ADO.NET.

(3) DataView

Once data stored into DataTable object, developers can use a **DataView object** to **view the data** in **different ways**. ADO.NET DataView objects allow developers **filter, sort, and search** the contents of DataTable objects, but they are **not SQL queries**. Developers cannot use a DataView to join data between two DataTable objects, nor can developers use a DataView to view only certain columns in a DataTable. DataView objects do **support filtering rows** based on **dynamic** in a **DataTable**. DataView objects do support **filtering** rows based on **dynamic criteria**, but they can access only a single DataTable, and all columns in the DataTable are available through the DataView.

Developers can also **locate** a DataRow in a DataTable based in the row's **primary key values**. The **Find method** is used to perform a **search** of the contents of the DataTable based on the **primary key value**. Although the Find Method is designed for DataTable objects, it is actually exposed by the DataRowCollection class. The Find method accepts an object that contains the primary key value for the row you want to locate. Because primary key values are unique, the find method can return at most one DataRow. The DataTable object's **Select method** is **powerful** and **flexible**, but it is not always the best solution.

(4) DataRelation

DataSet class defines a Relations property, which is a collection of DataRelation objects. A **DataRelation object** to indicate a **relationship** between **different DataTable** object in DataSet. The other functions of DataRelation object to **enforce constraints on the related DataTable**

objects. Related with Hierarchical XML Schemas, every implicit relationship between parent table and child table (see section 2.2) becomes a DataRelation in ADO.NET by childColumns or parentColumns. In addition, the related with Flat XML Schemas, every key ref definition (see section 2.2 (8)) becomes a DataRelation in ADO.NET.

2.5 The Sustaining Empirical Literature

There are several prior researches related with the conducting research. One of researches has been conducted by Leung (1987) on “**From a NIAM Conceptual Schema into the Optional SQL Relational Database Schema**”. The focus of studies is to know how to transform a well-formed NIAM conceptual schema into an SQL Optimal Normal Form (ONF) database schema. Relations in an ONF schema are at least in the fifth Normal form. The NIAM conceptual schema is selected to describe the UoD as it fulfills all the requirements. On the other hand, SQL (a relational language) is selected for describing the internal schema due to the fact it has become a de-facto standard for relational language.

Besides, the research above, Becker (1998) illustrates the similar concept in his studies on “**Normalization and ORM**”. He mentions that ORM is vastly superior to any other modeling technique at the conceptual level. In addition, it is also rich constraints and intuitive approach leading to better models. At the logical level, ORM still prevails as it generates equivalent fully normalized schemas without ever wondering what a functional dependency was. In his paper, he proves using external uniqueness constraint to solve the problem of Boyce-Codd Normal Form without losing a business rule.

As well, Suciu (2001) conducts a study on “**On Database Theory and XML**”. Suciu says over the years the connection between database theory and database practice

has weakened. She argues that the new challenges posed by XML and its application are strengthening this connection today. XML's semistructured data model represents paradigm shift for theoretical database research to practical database research.

The empirical literature on the purpose of Document Type Definition (DTD) is **“On XML Integrity Constraints in the Presence of DTDs”** has been conducted by Fan and Leonid (2001). They investigate XML documents specifications with DTDs and integrity constraints, such as keys and foreign keys. They have studied the consistency problem of checking whether a given specification is meaningful: that is, whether there exist an XML document that both conforms to the DTD and satisfies the constraints. They show that DTDs interact with constraint in a highly intricate way and as a result, the consistency problem in general is undesirable. When it comes to unary keys and foreign keys, the consistency problem is shown to be NP-complete. This is done by coding DTDs and integrity constraints with linear constraints on the integers.

Another empirical literature on the purpose of DTD is **“A Normal Form for XML Documents”** conducted by Arenas and Leonid (2002). XML documents may contain redundant information, and may be prone to update anomalies. Furthermore, such problems are caused by certain functional dependencies among paths in the documents. Their goal is to find a way of converting an arbitrary DTD into a well-designed one that avoids these problems. They first introduce the concept of a functional dependency for XML, and define its semantics via a relational representation of XML. They then define an XML normal form that avoids update anomalies and redundancies. They study its properties and show that it generalizes BCNF and a normal form for nested relations when those are appropriately coded as XML documents. Finally, they present a lossless algorithm for converting any DTD into XML normal form.

The other approach has been conducted by Arenas and Leonid (2003) in their paper with the title “**An Information-Theoretic Approach to Normal Forms for Relational and XML Data**”. Their goal is to provide a set of tools for testing when a condition on a database design, specified by a normal form, corresponds to a good design. They use techniques of information theory, and define a measure of information content of elements in a database with respect to a set of constraints. They first test this measure in the relational context, providing information-theoretic justification for familiar normal forms such as BCNF, 4NF, PJ/NF, 5NFR, DK/NF. They then show that the same measure is applied in the XML context, which gives us a characterization of a recently introduced XML normal form. They also propose normalization algorithm.

The interesting empirical literature on the purpose of XML Schema and Unified Modeling Language (UML) is “**UML and XML Schema**” conducted by Routledge, Linda, and Andrew (2002). They build on the approach by defining a mapping between the UML class diagram and XML Schema using the traditional three level database design approach (i.e. using conceptual, logical, and physical design levels). In their approach, the conceptual level is represented using the standard UML class notation, annotated with a few additional conceptual constraints. The logical level is represented in UML, using a set of UML stereotypes. And XML Schema itself represents the physical level. The goal of this three level design methodology is to allow conceptual level UML class models to be automatically mapped into the logical level, while minimizing redundancy and maximizing connectivity. They propose the methodology to mapping, logical level to physical levels, such as: (1) Generate type definitions, (2) Determine class groupings, (3) Build the complex type nesting, and (4) Create a root element.

The other researchers who work on XML Schema and UML is “**UML Documentation Support for XML Schema**” conducted by Salim et al. (2004). The research presents in their paper aims to: (1) assist understanding and documentation of XML Schema by converting them to graphical form, i.e. UML, and (2) to adopt a transformation approach that does not require additional training beyond standard UML.

2.6 The Essential Empirical Literature

This section discusses the most closely related works that involve XML Schemas and a conceptual framework NIAM, such as (Chankuang 2003) and (Chankuang 2004). Another research involves XML Schemas but uses Object Role Modeling (ORM) as a conceptual framework, conducted by Bird (2000). The researcher adopts their works with modification and revision related with the research that will be conducted.

First, Chankuang and Suphamit (2003) on “**A Software Tool for Object and XML Schemas Generation**” use a conceptual metaschema for design meta tables. The NIAM conceptual metaschema is a NIAM conceptual schema which describes components of a NIAM conceptual schema. Their software tool captures user’s conceptual schemas from the screen and populated the details of the conceptual schemas into the meta tables. Second, the same researchers, Chankuang and Suphamit (2004), on their paper with the title “**An Object and XML Database Schemas Design Tool**” use the NIAM conceptual schemas model as a conceptual framework for an XML Schema. Moreover, they transform from the NIAM schema to XML database schema. Furthermore, they introduce a software tool that allows users to enter the NIAM conceptual schemas and generate corresponding XML Schemas which guarantee minimum redundancy. They use the popular Supplier-Part database to explain the transformation, Date (2000), as shown in Figure 2.21. In Practice, a NIAM

conceptual schema will be created first. After that, it will be transformed into a corresponding XML schema. They propose the transformation steps are as follow:

- (A) Specify a root element name of the XML schema.
- (B) Each label type and leaf-node entity type is transformed into and XML simple type (see Figure 2.27.).

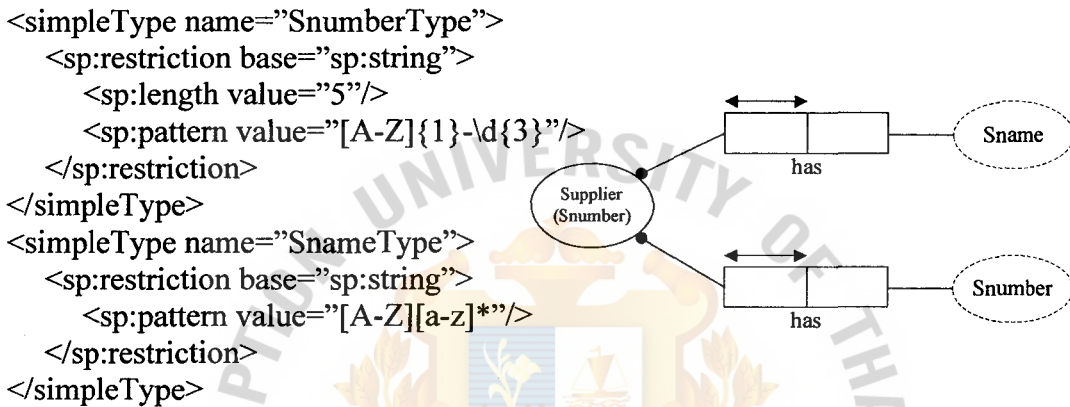


Figure 2.27. A Entity Type SUPPLIER, Its Value Types and Its Corresponding XML Schema.

- (C) Each main entity type is transformed into an XML complex type. Many-to-one and one-to-one binary relationship types that main entity type is involved transform into elements of the complex type. Each corresponding element name is either the unique identifier of participating entity types or label types (see Figure 2.28.).
- (D) Each binary many-to-many fact type is transformed into an XML complex type (see Figure 2.29.).
- (E) Each n-ary fact type is transformed into an XML complex type (see Figure 2.30.).

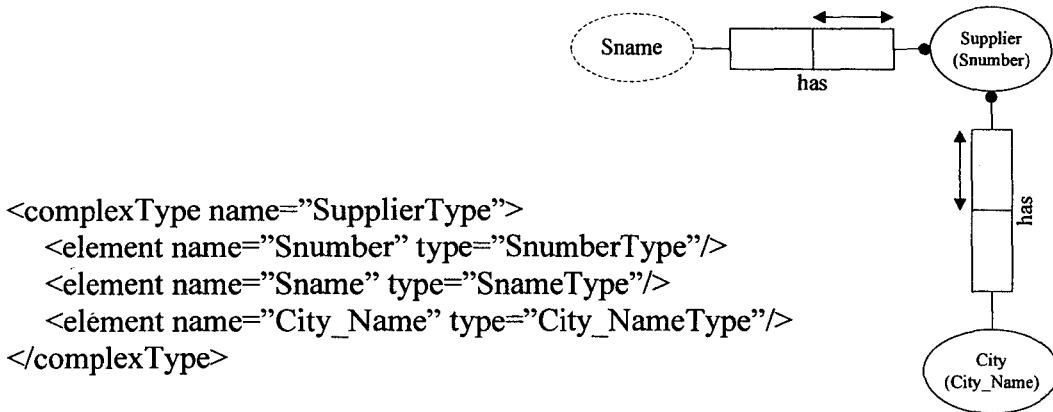


Figure 2.28. A NIAM Schema with Many-to-one Relationships and a Corresponding XML Schema.

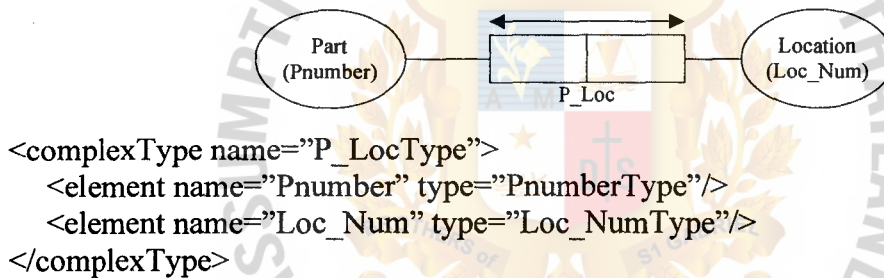


Figure 2.29. A NIAM Schema with Many-to-many Relationships and a Corresponding XML Schema.

- (F) The unique identifier of each main entity type is transformed into an XML schema key, see listing below:

```

<key name="SnumberKey">
  <selector>Supplier_Part/Supplier</selector>
  <field>./Snumber</field>
</key>

```

- (G) The uniqueness constraint on roles of a binary many-to-many fact type or an n-ary fact type is transformed into an XML unique constraint, see listing below:


```

<unique name="SupplyKey">
  <selector>Supplier_Part/Supplier</selector>
  <field>./Snumber</field>
  <field>./Pnumber</field>
</unique>

```

```

<complexType name="SupplyType">
  <element name="Snumber" type="SnumberType"/>
  <element name="Pnumber" type="PnumberType"/>
  <element name="No_of_Item" type="No_of_ItemType"/>
</complexType>

```

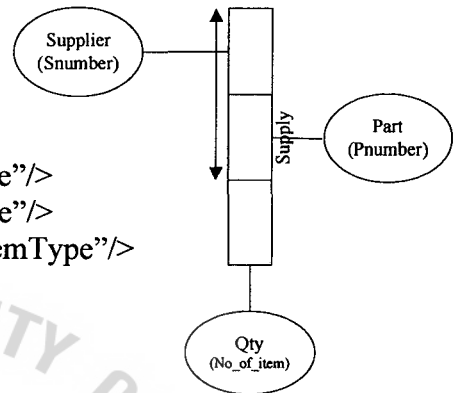


Figure 2.30. A NIAM Schema with a Ternary Fact Type and a Corresponding XML Schema.

- (H) Fact types between main entity types form XML schema key references, see listing below:

```

<keyref name="Supply_SupplierRef1" refer="SnumberKey">
  <selector>Supplier_Part/Supply</selector>
  <field>./Snumber</field>
</keyref>

<keyref name="Supply_SupplierRef2" refer="PnumberKey">
  <selector>Supplier_Part/Supply</selector>
  <field>./Pnumber</field>
</keyref>

```

Lastly, the other researcher who works in XML Schema although use ORM is **“Object Role Modeling and XML Schema”** conducted by Bird, Andrew, and Terry (2000). They investigate the use of ORM, a conceptual modeling method, as a mean for designing XML Schemas. By using ORM, they are able to model a rich variety of data constraints, and delay decisions about the tree-structure of the conceptual analysis

phase. They estimate that encoding an ORM schema in XML Schema has benefits beyond facilitating the exchange of schemas between different CASE tools and repositories. They propose algorithm for generating an XML Schema from ORM diagram: (1) Generate a type definition for each ORM object type, (2) Build a complex type definition for each major fact type grouping, and (3) Create a root element for the whole schema and add keys and key references.

2.7 Summary

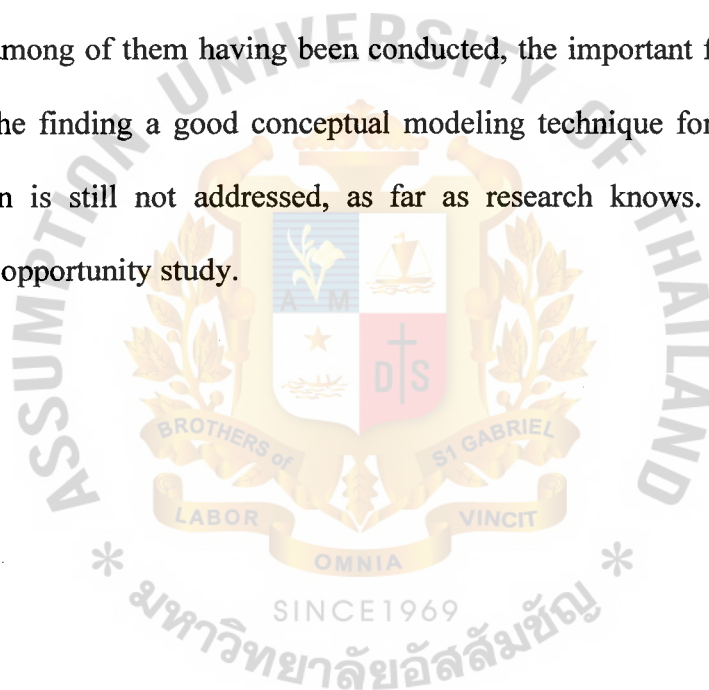
The literature review has provided an insight about various critical dimensions of NIAM, XML Schema, and XML Document. Leung has just studied how to transform a well-formed NIAM conceptual schema into an SQL ONF database schema. Additionally, Becker mentions the NIAM riches constraints and intuitive approaches lead to fully normalization schemas without ever wondering what a functional dependency is. The above researches are around theory database. Suciu argues that the new challenges posed by XML open connection between database theory and database practice research.

After W3C recommended XML, many researchers conduct researches in this area. Fan and Leonid conduct research integrity constraint and consistency XML documents with DTD. Furthermore, Arenas and Leonid study to convert DTD into a well-designed one in order to avoid update anomalies and redundancies. Moreover, Arenas and Leonid use techniques of information theory create XML normal form. The other researcher, Routledge et al., build on the approach by defining a mapping between the UML class diagram and XML Schema using conceptual, logical, and physical design levels. The last researchers, Salim et al., study to convert from XML Schema to graphical form, i.e. UML.

The most closely related works that involve XML Schemas and a conceptual

framework NIAM, firstly Chankuang and Suphamit use a conceptual metaschema for design meta tables. Halpin (1995) mentions there are many possible solutions which produce difference metaschema. Researchers use metaschema to reengineering XML Schema therefore the metaschema should modify. In another paper, they use the NIAM conceptual schemas model as a conceptual framework for an XML Schema. Furthermore, they transform from the NIAM schema to XML database schema. The researcher will adopt with modifying the transformation one-to-one relationship.

Despite many researches related with XML, ORM, NIAM, normalization or combination among of them having been conducted, the important fundamental issues related with the finding a good conceptual modeling technique for XML conceptual schema design is still not addressed, as far as research knows. Therefore it is a challenge and opportunity study.



III. RESEARCH METHODOLOGY

3.1 Determine and Define the Thesis Theme

The researcher explores the empirical literatures from the international proceedings and the international journals in order to determine what conceptual schemas have been and have not been conducted by the prior researches. In addition, to support conducting conceptual framework, the researcher explores the theoretical literatures from library and Internet, and finally defines the theme for the thesis.

One remarkable issue remaining uninvestigated is the applying NIAM conceptual schema for improving poorly designed the XML Schema to be the normalized XML Schema and the normalized XML Document. Therefore, the researcher defines the thesis title “**A Conceptual Schema Approach for XML Schema Design**”. Apply NIAM conceptual schema to reengineering XML Schema.

3.2 Research Method

The research type conducted is an applied research. The aims of the thesis are to find a good conceptual modeling technique for XML Schema and to create software tool for reengineering XML Schemas using the NIAM conceptual schema. The research is conducted by building a model (software tool) and a simulation. Methods for construction of software tools are creating algorithms and generating software tools.

Once the software tool is built, tested and validated the model faithfully captures the relevant aspects of the modeled system. Finally, simulation experiments are tested to prove that software tools have already been free from bugging.

3.3 Research Design

A conceptual framework for developing software tool in this research is shown at Figure 3.1. In this study, the researcher assumes that the XML Schema and the XML

Step by step for developing application tools are:

- (1) Identify Hierarchical Data Model and Hierarchical XML.
- (2) Identify NIAM Conceptual Schema and XML Conceptual Schema.
- (3) Create the NIAM Conceptual metaschema.
- (4) Create the transforming from Hierarchical XML into Flat XML algorithms.
- (5) Generate the reverse engineering from XML Schemas into NIAM conceptual schemas algorithms. This algorithm applies NIAM CSDP.
- (6) Create the forward engineering from NIAM conceptual schemas into XML Schemas algorithms. This algorithm applies NIAM Rmap.
- (7) Create the writing Flat XML Schemas algorithms.
- (8) Create the writing Flat XML Document algorithms.
- (9) Build a software tool for implementing all of algorithms with Microsoft Visual Basic .NET. The built software tool must have ability to evaluate and revise the XML Schema and the XML document using NIAM conceptual schema to become optimal normal.
- (10) Test and validate the building software tool.
- (11) Perform simulation experiments to test that software tool has already been free from bug. For example use XML Writer to check and validate XML Schema and XML Document created are well-formatted and validated.

IV. SYSTEM DEVELOPMENT

This chapter discusses about comparison Hierarchical Data Model and Hierarchical XML. Additionally, the chapter discusses about comparison NIAM conceptual schema and XML conceptual schema. Moreover, the chapter presents the NIAM Conceptual metaschema which meta tables will be used to record the NIAM conceptual schema in the created software tool. Furthermore, the chapter presents the overview and demonstrates algorithm by example for transforming from Hierarchical XML into Flat XML algorithms, reverse engineering from XML Schemas into NIAM conceptual schemas algorithms, and forward engineering from NIAM conceptual schemas into XML Schemas algorithms. Finally, the chapter presents input and output design for software tool.

4.1 Comparison Hierarchical Data Model and Hierarchical XML

In this section, the researcher compares between Hierarchical Data Model (see section 2.1) and Hierarchical XML in order to analyze a Hierarchical XML (see section 2.2) and proposes several alternative solutions for handling the Hierarchical XML. The research must comprehend the theoretical Hierarchical Data Model and the hierarchical, which are implemented in XML. There are similarities and differences between them. To identify XML data structures, the researcher explores two examples Hierarchical XML Schema and XML Document, Mortgage and Suppliers Parts.

The first similarity between Hierarchical Model Data and Hierarchical XML, both of them can be displayed as a **hierarchical diagram**. For example, Figure 2.2. shows a hierarchical diagram for the hierarchical schema of COMPANY Database in Hierarchical Model Data. Similarly, Hierarchical Mortgage XML Schema can be displayed as a hierarchical diagram (see Figure 2.9.).

The second similarity between Hierarchical Model Data and Hierarchical XML, both of them use **hierarchical sequence** in order to produce a sequence of record occurrence tree, **record type indicator** (see section 2.1 (4)). To identify the XML Schemas, the researcher uses ADO.NET as tools. ADO.NET provides a result as shown in Figure 4.1. for Figure A.1. The table shows additional column, it calls **record type indicator**. In ADO.NET, it calls **ChildColumns** or **ParentColumns**. It is the same idea with Ramakrishnan (2003) (see the last section 2.2). The **record type indicator** writes with bold font style in Figure 4.1. For example, column **document_Id** is added into table Document, Mortgagee, Mortgage, and Bank. To better understand with hierarchical sequence, the researcher uses ADO.NET once more to overview Figure A.2. The result appears in 4.2. Equally, the researcher identifies Figure A.5., the result appears in Figure 4.3. Additionally, the researcher explores Figure A.6., see Figure 4.4. as a result.

Column Name	Column Type
documentDate	date
comment	string
document_Id	Int32

(a) Document Table

Column Name	Column Type
name	string
location	string
city	string
state	string
phone	string
document_Id	Int32

(b) Mortgagee Table

Column Name	Column Type
property	string
date	date
loanAmout	decimal
term	integer
loanNumber	string
document_Id	Int32

(c) Mortgage Table

Column Name	Column Type
name	string
location	string
city	string
state	string
phone	string
document_ID	Int32

(d) Bank Table

Figure 4.1. Hierarchical Mortgage XML Schema in ADO.NET.

documentDate	comment	document Id
2005-03-02	Good risk	0
2004-07-11	Good	1
2004-07-14	Good	2

(a) Document Table

name	location	city	state	phone	document Id
James Blandings	1234 299th St	New York	NY	888.555.1234	0
Hans Schmidt	123 Hallgarten	Berlin		870.220.5678	1
Hans Schmidt	123 Hallgarten	Berlin		870.220.5678	2

(b) Mortgagee Table

Property	date	loanAmount	term	loanNumber	document Id
The Hackett Place	2005-03-01	80000	15	66 7777 88	0
123 Acorn Drive	2005-03-01	90000	15	11 8888 22	0
99 West Pocusset St	2005-03-02	100000	30	33 4444 11	0
19 Johnson Place	2005-03-02	110000	30	55 3333 88	0
345 Nottingham Court	2005-03-02	120000	30	22 6666 99	0
Vila	2004-07-12	55000	12	11 2233 44	1
House	2004-12-25	95000	24	11 1222 33	1
Bungalow	2004-07-12	50000	12	12 3122 34	2

(c) Mortgage Table

name	location	City	state	phone	document Id
XML Bank	12 Schema Place	New York	NY	888.555.8888	0
Niaga	56 Sweet Street	Berlin		811.110.1234	1
XML Bank	12 Schema Place	New York	NY	888.555.8888	2

(d) Bank Table

Figure 4.2. Hierarchical Mortgage XML Document in ADO.NET.

The third similarity between Hierarchical Model Data and Hierarchical XML, both of them allow **duplication data** of child record instance to implement **many-to-many relationship** (see section 2.1 (2)). For example, Figure 4.4. (b) shows Pnumber, “P1” appears two times, “P2” appears four times, “P4” appears two times, and “P5” appears two times.

Column Name	Column Type
Snumber	string
Sname	string
Status Num	string
City Name	string
Supplier_Id	Int32

(a) Supplier Table

Column Name	Column Type
Pnumber	string
Pname	string
ColorName	string
Loc Num	string
No of Item	integer
Supplier_Id	Int32

(b) Part Table

Figure 4.3. Hierarchical Supplier-Part-A XML Schema in ADO.NET.

Snumber	Sname	Status Num	City Name	Supplier_Id
S1	Smith	20	London	0
S2	Jones	10	Paris	1
S3	Blake	10	Paris	3
S4	Clark	20	London	4
S5	Adams	30	Athens	Null

(a) Supplier Table

Pnumber	Pname	ColorName	Loc Num	No of Item	Supplier_Id
P1	Nut	Red	London	300	0
P2	Bolt	Green	Paris	200	0
P3	Screw	Blue	Rome	400	0
P4	Screw	Red	London	200	0
P5	Cam	Blue	Paris	100	0
P6	Cog	Red	London	100	0
P1	Nut	Red	London	300	1
P2	Bolt	Green	Paris	400	1
P2	Bolt	Green	Paris	200	3
P2	Bolt	Green	Paris	200	4
P4	Screw	Red	London	300	4
P5	Cam	Blue	Paris	400	4

(b) Part Table

Figure 4.4. Hierarchical Supplier-Part-A XML Document in ADO.NET.

The first difference between Hierarchical Model Data and Hierarchical XML is in **hierarchical relationship definition**. Naturally, in Hierarchical Model Data, a PCR type defines a hierarchical **one-to-many** relationship between parent record type and child record type (see Section 2.1 (1)). For example, Figure 4.5. (c) shows DUCUMENT as parent record type and MORTGAGEE, MORTGAGE, and BANK as child record type. By PCR type definition, it means that one DUCUMENT record type has many MORTGAGEE record types and also one DUCUMENT record type has many BANK record types. Nevertheless, those do not appropriate with a case problem (see preface chapter 2). While DUCUMENT record type has many MORTGAGE record types is right by PCR type definition.

However, in XML Schemas, the implementation of PCR type does not always define a hierarchical one-to-many relationship between main-element and sub-element. It depends on **maxOccurs** that defines at element in XML Schema and duplication or induplication instance in sub-element. For example, Figure 4.5. (a) shows that all element definitions in the **documentType** complex type, both **minOccurs** and **maxOccurs** are **one**, although they do not write them down (see convention in section 2.2 (5)). The **minOccurs** and the **maxOccurs** are presented by **(minOccurs:maxOccurs)**, for illustration see Figure 4.5. (b). In XML Schemas, a sub-element does not always as a child record type (see explanation in the next paragraph).

The second difference between Hierarchical Model Data and Hierarchical XML is in the **purpose of duplication child record instance**. In Hierarchical Model Data, duplication instance in child record type is to implement many-to-many relationship (see section 2.1 (2)). In Hierarchical XML duplication child record instance do not only to implement many-to-many relationship but also to implement many-to-one relationship. However many-to-one does not discuss in Hierarchical Model Data. For

example, see the definition of mortgagee in Figure 4.5. (a) line 17, max Occurs is 1 and see the instance in Figure 4.2. (b), "**Hans Schmidt**" is recorded double. Similar with bank, see the definition of mortgagee in Figure 4.5. (a) line 19, max Occurs is 1 and see the instance in Figure 4.2. (d), "**XML Bank**" is recorded double. The duplication is not wrong because they relate with different document. From that example the researcher briefs there are duplications in many-to-one. In this case the sub-element becomes the parent of record type. Furthermore, Duplication instance in sub-element is symbolist with positive sign (+) and induplication is symbolist with negative sign (-), for example see Figure 4.5. (b).

The third difference between Hierarchical Model Data and Hierarchical XML is in implement **many-to-many relationship** between parent record type and child record type. Elmasri's suggestion is shown in Figure 4.6. (a). Based on literature review, the researcher observes all authors and examples in Schema Part 0 (primer) write XML Schema like Figure 4.6. (b). It is estimated that they just give an example how to create and to display one instance in XML Document. However, the XML file in this research is used to transfer database, therefore the researcher implements many-to-many relationship of Elmasri solution (see Figure 4.6. (a)) in software tool.

The researcher tries to define maxOccurs directly in element as the first propose solution. For example see Figure 4.7. (a) line 15. See Figure A.5. and Figure A.6. respectively for full listing XML Schema and XML Document. The researcher defines minOccurs="0" maxOccurs="unbounded" directly in element **Part** in main-element **Supplier**. Therefore **Part** becomes a sub-element of **Supplier**. On the other hand, there is also limitation editor (see section 2.2 (9)). The researcher proposes another solution for many-to-many relation as the second proposed solution, especially while the middle node has the other element except **record type indicator** (see Figure 4.6. (c)). The

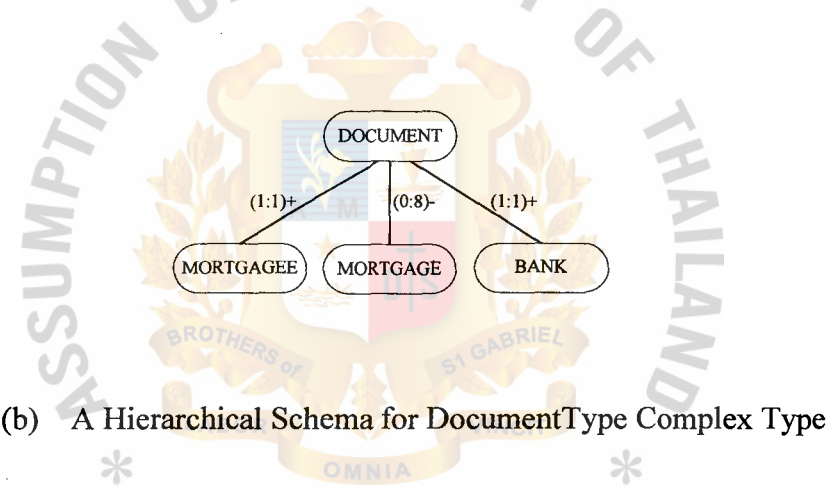
second proposed XML Schema and XML Document is listed in Figure A.7. and Figure A.8. respectively.

```

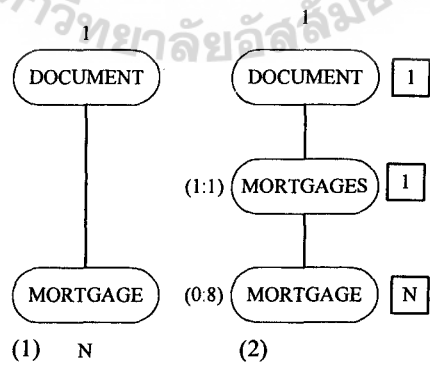
14 <xsd:complexType name="documentType">
15   <xsd:sequence>
16     <xsd:element ref="comment" minOccurs="1" />
17     <xsd:element name="mortgagee" type="recordType" /> ← (1:1)
18     <xsd:element name="mortgage" type="mortgagesType" minOccurs="0"
19       maxOccurs="8"/> ← (0:8)
20   </xsd:sequence>
21   <xsd:attribute name="documentDate" type="xsd:date" />
22 </xsd:complexType>

```

(a) documentType Complex Type of Mortgage XML Schema in Figure A.1.

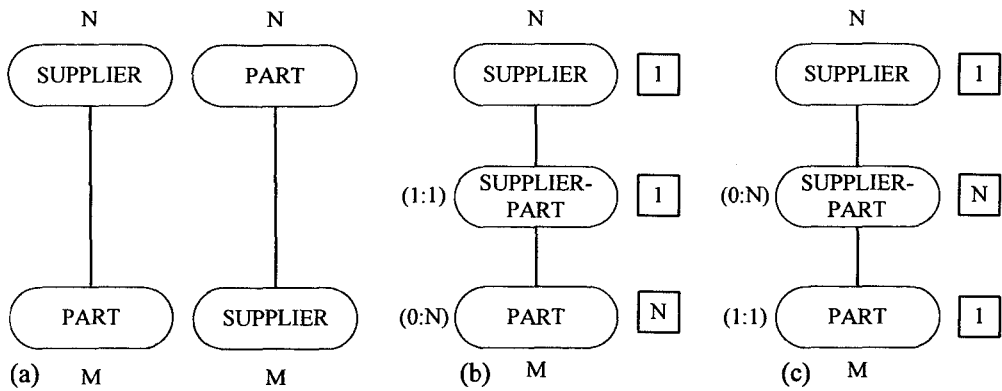


(b) A Hierarchical Schema for DocumentType Complex Type



(c) A Tree Representation for DocumentType Complex Type

Figure 4.5. Implemented PCR type in Mortgage XML Schema (DocumentType Complex Type).



- (a) Elmasri's Solution for Many-to-Many Relationship
- (b) Implementation Many-to-Many in XML Schema
- (c) Alternative Many-to-Many Solution in XML Schema

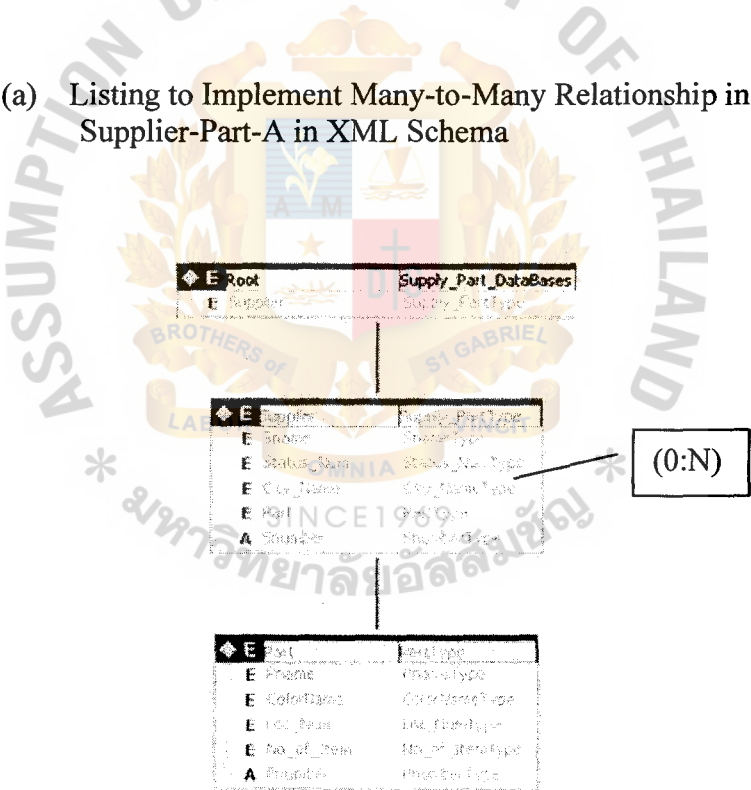
Figure 4.6. Comparison the Many-to-Many Relationship in Hierarchical Schema.

In short, those are similarity between Hierarchical Data Model and Hierarchical XML, in hierarchical diagram appearance, hierarchical sequence, and duplication data of child record instance to implement many-to-many relationship. In addition, those are also differentiates between Hierarchical Data Model and Hierarchical XML in hierarchical relationship definition, the purpose of duplication child record instance, and implementation many-to-many relationship. The key points from this section is maxOccurs, it plays important role in hierarchical relationship definition. If maxOccurs is declared more than one and no duplication in sub element it means one-to-many relationship. On the other of hand, it means many-to-many relationship. In addition, while maxOccurs is declared one and no duplication in sub element it means one-to-one relationship. Duplication in sub element, however, it means many-to-one relationship. The knowledge from this section is used to transform from Hierarchical

XML Schemas into Flat XML Schemas algorithms.

```
4      <sp:complexType name="Supply_Part_DataBases">
5        <sp:sequence>
6          <sp:element name="Supplier" type="Supply_PartType"
maxOccurs="unbounded" />
7        </sp:sequence>
8      </sp:complexType>
9      <sp:complexType name="Supply_PartType">
10        . . .
14      <sp:element name="Part" type="PartType" minOccurs="0"
maxOccurs="unbounded" />
15    </sp:sequence>
16    <sp:attribute name="Snumber" type="xsd:string " />
17  </sp:complexType>
18  <sp:complexType name="PartType">
19    . . .
26  </sp:complexType>
```

(a) Listing to Implement Many-to-Many Relationship in Supplier-Part-A in XML Schema



(b) Hierarchical Diagram to Show the Implement Many-to-Many Relationship in Supplier-Part-A XML Schema

Figure 4.7. Implementation Many-to-Many Relationship in Supplier-Part-A Hierarchical XML Schema.

4.2 Comparison the NIAM Conceptual Schema and the XML Conceptual Schema

This section discusses a comparison between the NIAM conceptual schema and the XML conceptual schema in order to answer the research question “**Can the NIAM CSDP and the NIAM Rmap apply meaningfully to improve the poorly designed XML Schema?**” From literatures review, the researcher acquires knowledge about NIAM conceptual schema and XML. The NIAM conceptual schema comprises three main sections, i.e.: stored fact types, constraints, and derivation rules. There are similarities and difference between NIAM conceptual schema and XML conceptual schema. The similarities are in stored fact types and in constraints and differentiation is in derivation rules.

First, the similarity in stored fact types, a main entity type (see section 2.3 (3)) in the NIAM is similar with a complex type in the XML. In addition, a label type or value type in the NIAM is analogous with an element XML. Furthermore, a main entity type comprises several label types or value types similar to a complex type. A complex type consists of several elements.

Second, the similarity in constraints, validation rules or integrity rules in the NIAM is implemented with uniqueness constraint and reference type (see section 2.3.). In XML, validation rules or integrity rules are applied with key and keyref (see section 2.2 (8)). There are correspondences between uniqueness constraint and key and between reference type and keyref. Additionally, another constraint, lists various constraint or restriction that apply to population of the NIAM fact type is like data facet in simple type. The example how to define value constraints in XML is discussed in section 2.2 (10). The other important constraints beside key and keyref are minOccurs and maxOccurs. minOccurs and maxOccurs in the NIAM is useful in mandatory roles.

Related with XML, minOccurs and maxOccurs are useful in content models.

Lastly in derivation rules, although both NIAM and XML have derivation rules, they are used for different function. In NIAM, derivation rules provide a list of function, operators and rules that may be used to derive information not explicitly stored in database. These may involve mathematical calculation or logical inference. In XML, on the other hand, derivation rules are used to deriving types by extension, deriving complex type by restriction.

In short, from the discussion above, the NIAM conceptual schema generally can be applied to improve the poorly designed the XML conceptual schema, except derivation role. It is not too significant. Therefore in step two CSDP, check any arithmetic derivations, should not apply in reverse engineering algorithms.

4.3 The NIAM Conceptual Metaschema

In section 4.2, the researcher compares between the NIAM and the XML conceptual schema. Therefore, the researcher knows what the NIAM sections can and can not be applied in the XML conceptual schema. This section observes stored fact types and constraints in ADO.NET as a tool in the study. Moreover, the section presents the NIAM conceptual metaschema with the aim of creating meta tables. The meta table will be used in the research to record fact type in reverse engineering process and to map back the NIAM conceptual schema into the relational schemas, XML Schema.

Section 2.4 (2) discussing every complex type in XML Schemas becomes a DataTable in ADO.NET and every element in complex type becomes a DataColumn of DataTable in ADO.NET. This section continues the discussion with the topic of interest, property. The researcher compares between properties attribute in XML (see Table 2.1.) and DataColumn ADO.NET (see Table 2.7.) aiming to store fact types.

Based on two tables, the researcher examines there are several Attributes in XML, which can be captured in DataColumn ADO.NET but quite a lot can not be captured. The XML Attributes that can be captured in DataColumn ADO.NET are default, id, fixed, name, nillable, and type. A correspondence between the captured XML attributes name and properties name of DataColumn ADO.NET is shown in Table 4.1. The XML Attributes, however, unable to implement in DataColumn ADO.NET are abstract, block, final, maxOccurs, minOccurs, ref, substitutionGroup, and use.

Table 4.1. The Implemented XML Attributes in DataColumn Properties ADO.NET.

XML Attribute Name	Properties Name DataColumn ADO.NET
default	DefaultValue
id	AutoIncrement, AutoIncrementSeed, AutoIncrementStep
fixed	ReadOnly
name	ColumnName
nillable	AllowDBNull, minOccurs
type	DataType

Furthermore, the researcher also compares the XML facet element (see Table 2.2.) with properties DataColumn ADO.NET (see Table 2.7.). Based on two tables, the researcher examines there is only one facet element in XML which can be implemented in DataColumn ADO.NET but quite a lot can not be captured. The XML facet element that can be captured in DataColumn ADO.NET is maxLength only, with the sama name. The XML facet element, however, can not be captured in DataColumn ADO.NET are enumeration, fractionDigits, length, maxExclusive, maxInclusive,

minExclusive, minInclusive, minLength, pattern, totalDigits, and whiteSpace.

The researcher discusses in section 4.2, how important the constraints minOccurs and maxOccurs, beside key and keyref, in the NIAM and in the XML. In addition, there are many XML attribute and XML facet element can be not applied in DataColumn ADO.NET. In short, if XmlSchemaRead which is one of ADO.NET methods is used to read the schema into DataSet than a lot XML attributes and XML facets will be lost. Although ADO.NET is powerful enough in conversion XML into database and vice versa, data sort, data filter but it does not full support XML. To capture XML attribute and XML facet element that can be not applied in DataColumn ADO.NET, it needs use XmlTextReader method.

Furthermore, the researcher designs a NIAM conceptual metaschema to record stored fact types and constraints. Figure 4.8. shows the created NIAM conceptual metaschema. Later, the researcher transforms the NIAM conceptual metaschema to meta tables that will be used in designing software tool in next section. As additional note, meta tables are saved in **myDS**. Figure 4.9. appears the created meta tables. Meta tables consist of two categories. First category is used to record XML Schema, such as SysComplex, SysComplexElement, RelationRef, and Attribute-Facet. The second category is used to record the NIAM conceptual schema of reengineering XML Schema, such as Object, Fact Type, and Role.

One of category meta tables to record XML Schema is **SysComplex** Table. The table is used to record every complex type in XML Schemas. The contents of SysComplex Table are complex type name (as primary key), number of element in the complex type, and key of complex type. In addition, **SysComplexElement** Table is used to save every element in the complex type. The table consists of code element (as a primary key), name of complex type, element name, and label type. Label type will

be used as reference to Object Table, to combine entity type in CSDP third step. Moreover, **RelationRef** Table is used to record the table name that will be related (as primary key), the parent table name, the primary key of parent table, the child table name, and the primary key of child table. The other purpose of **RelationRef** Table is to record documentation of the different column names from parent table and child table, which have the same entity type. Lastly, **Attribute-Facet** Table is used to record attribute elements and facets that are not covered in ADO.NET, except minOccurs. It will be recorded in **Role** table. Moreover, only more than one maxOccurs should be saved in the table. With assumption, if maxOccurs does not exist in **Attribute-Facet** Table is one, the table consists of code element and code attribute or facet (as primary keys), and value of attribute or facet.

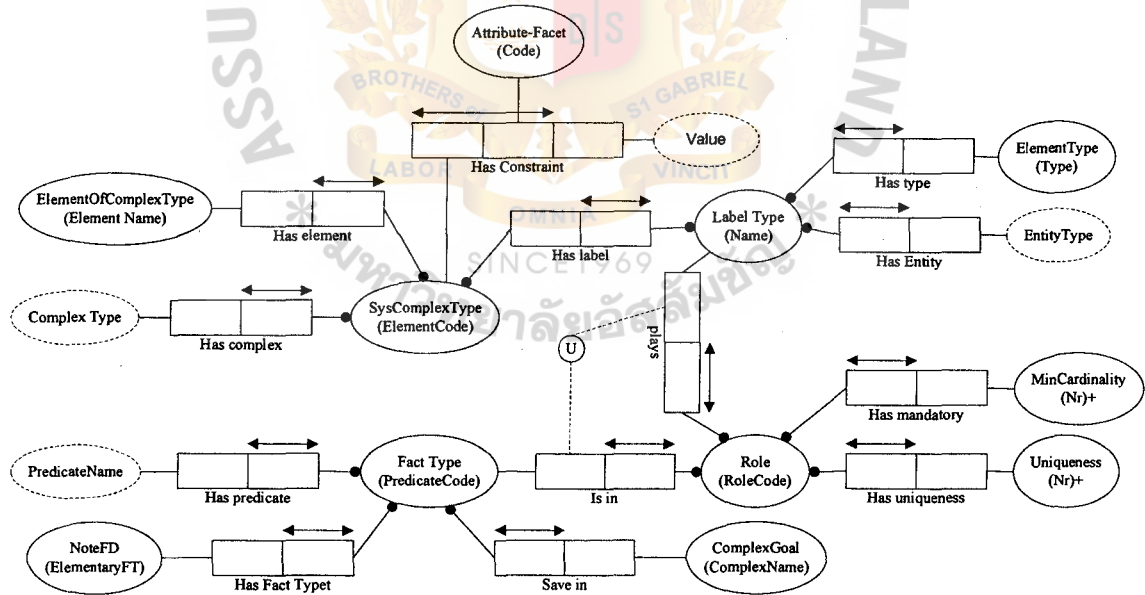


Figure 4.8. The NIAM Conceptual Metaschema.

Coding for **Attribute-Facet** Table are abstract → 1, block → 2, final → 3, maxOccurs → 4, ref → 5, substitutionGroup → 6, use → 7, enumeration → 8, fractionDigits → 9, length → 10, maxExclusive → 11, maxInclusive → 12, minExclusive → 13, minInclusive → 14, minLength → 15, pattern → 16, totalDigits → 17, and whiteSpace → 18.

Column Name	Column Type
ComplexType	string
NumberOfColumns	string
PrimaryKey	string

(a) SysComplex Table

Column Name	Column Type
EntityType	string
LabelType	string
ElementType	string

(e) Object Table

Column Name	Column Type
ElementCode	integer
ComplexType	string
ElementOfComplexType	string
LabelType	string

(b) SysComplexElement Table

Column Name	Column Type
PredicateCode	string
PredicateName	string
NoteFD	string
ComplexGoal	string

(f) Fact Type Table

Column Name	Column Type
Relation	string
ParentComplex	string
ParentKey	string
ChildComplex	string
ChildKey	string

(c) RelationRef Table

Column Name	Column Type
RoleCode	string
PredicateCode	string
LabelType	string
Uniqueness	integer
MinCardinality	integer

(g) Role Table

Column Name	Column Type
ElementCode	integer
Code	integer
Value	string

(d) Attribute-Facet Table

Figure 4.9. The Meta Tables.

One of category meta tables to record the NIAM conceptual schema is **Object Table**. The table is used to record object type, such as entity type name, label type name (as primary key), and data type of label type. Another table is **Fact Type Table**. The table is used to save NIAM fact type of reengineering XML Schema. The table comprises of predicate code (as primary key), predicate name, functional dependency note, and complex type name (for forward engineering). The most important table is **Role Table**. The purpose table is used to record documentation of the role that will be played by every entity type or value type, uniqueness constraints and mandatory role constraints. The table consists of role code, predicate code and element code as primary keys, uniqueness, and minimum cardinality (minOccurs).

4.4 The Transforming from Hierarchical XML into Flat XML Algorithms

Duplication instances in the sub-element in Hierarchical XML cause problems of wasting storage and maintaining consistent. In addition, at present, most companies apply relational database application, as reference see the homepage <http://www.rpbouret.com/xml/XMLDatabaseProds.htm>. Furthermore, Elmasri (1994) says that in general the hierarchical model works well for database applications which are naturally hierarchical. However, when there are many nonhierarchical relationships, trying to fit those relationships into a hierarchical form is then difficult. Also the results are often unsatisfactory. The researcher agrees with Elmasri's opinion, therefore the the algorithm is proposed to transform the Hierarchical XML into Flat XML one in this section.

The transformation from Hierarchical XML into Flat XML algorithms will be separated into two major steps, i.e. transforming from Hierarchical XML Schemas into Flat XML Schemas algorithms and converting from Hierarchical XML Documents into Flat XML Documents algorithms. In general, the Hierarchical to Flat transformation

algorithm will start from level 0 (root table) to one level before the last level and from left to right child table. In this study, a processed table in the present level is called parent table. For example, see Figure 4.10. the top left, assume the present level is level 1. As a parent table is CHILD12. A parent of CHILD12 is ROOT which is called grand parent and a child of CHILD12 is CHILD21 which is called child.

The first major step of transformation from Hierarchical XML into Flat XML algorithms is transforming from Hierarchical XML Schema into Flat XML Schema. To make easily inclusive about the related XML Schema, ADO.NET, and the transforming hierarchical XML schema into flat XML schema algorithm, see Figure 4.10. For the next discussion, the researcher draws simple block diagram as shown in the bottom of the diagram block. The input of the process is Hierarchical XML Schemas and the outputs of the process are **DocumentDS** and **myDS**. **DocumentDS** is used to record the created Flat XML.

The algorithms to transform from Hierarchical XML Schemas into Flat XML Schemas are:

Step 1: Read XML schema using **ReadXmlSchema** method into **OldSchemaDS**.

Then load XML document into **OldDocumentDS** using **ReadXml** method. In addition, record every attribute and facet into Attribute-Facet Table. Finally, create a **RelationRef** table (see section 4.3).

Step2: In order to transform implicit to explicit relationship between element and sub-element, acquire primary key for every complex type from the user. Follow by save a parent table name in the level 0 into **ParentNodeArr** array.

Step3: Check the content **ParentNodeArr** array because the content in the bottom array will be used as the processed parent table in the present level. Moreover, it will be used as a condition loop, if the array does not empty then do **step 4**

```
else terminate process.
```

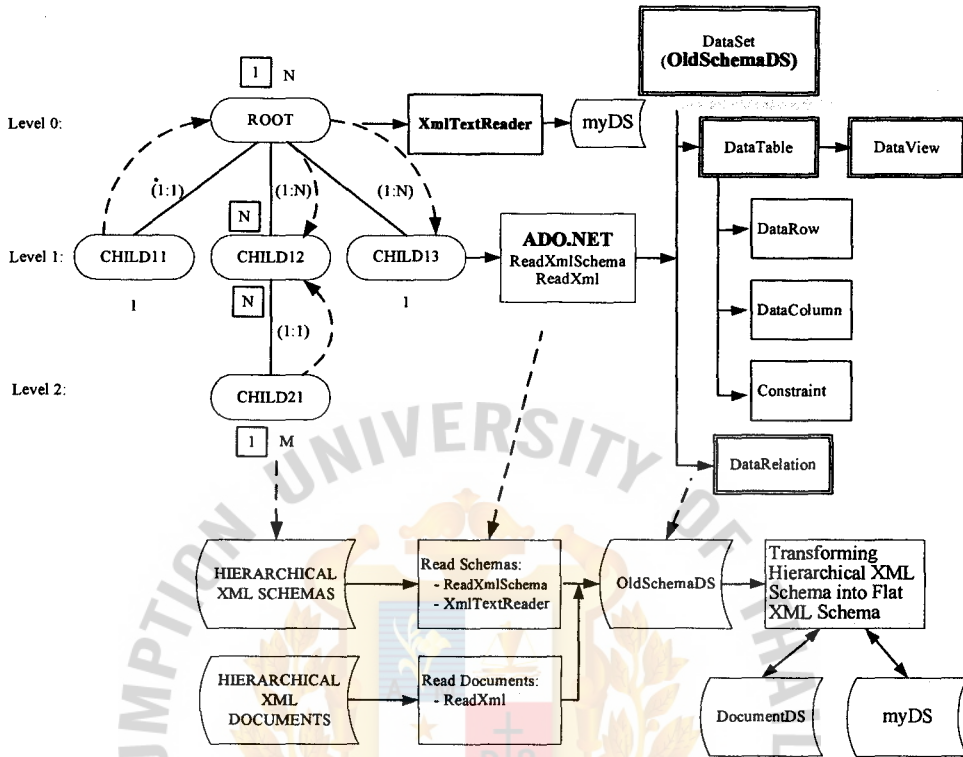


Figure 4.10. Transforming Hierarchical XML Schemas into Flat XML Schemas Block Diagram.

Step 4: Read the content at the bottom of **ParentNodeArr** array, as a parent table name. Then read the parent table schemas from **OldSchemaDS**. Furthermore, create a new table in **DocumentDS** with the same name and schema as parent table schemas in **OldSchemaDS**, except **ChildColumns** or **ParentColumns** not included. Finally, set the **primary key** on the new table schema with the primary key parent table in **OldSchema**.

Step 5: Get every child table of the parent table in the next level. There are two possibility statuses of the child table, as a leaf table or not as a leaf table. Therefore, this step should check the status of the child table. Furthermore,

process every child from the left child table until the end of right child table through step 6 until step 11.

Step 6: If the child table is a leaf table, then read the child table schemas from **OldSchema** and create new table with the same name and schema in **DocumentDS**, except ParentColumns. After that, set the **primary key** on the new table schemas as the primary key child table on the **OldSchema**. Afterward, check the **maxOccurs** of the child table.

Step 7: If the child table is a leaf table and the **maxOccurs** of the child table is defined one then save the primary key new table into parent table as **foreign key**. As well, the program must check the column name in the parent table because the same column name can not exist in a table. For that purpose and mapping relationship between, **RelationRef** table (see section 4.3) is used, then go to **Step 11**.

Step 8: If the child table is a leaf, the maxOccurs of the child table is defined more than one and duplication instance in the child table then save the primary key of parent table into the new table as part of the primary key of the new table, then go to **Step 11**.

Step 9: If the child table is a leaf, the maxOccurs of the child table is defined more than one and induplication instance in the child table then save the primary key of parent table into the new table as a foreign key of the new table, then go to **Step 11**.

Step 10: If the child table is not a leaf table then only save the name of the child table into **ParentNodeArr** for next process.

Step 11: Save the relationship between the parent table and the child table in **RelationRef** table.

Step 12: After all child table have already been processed then delete the bottom content of **ParentNodeArr**. Next, go to **Step 3**.

The researcher demonstrates the algorithm to Mortgage information, which XML Schema created by Holzner. In the first step, Programs read XML schema using ReadXmlSchema method into **OldSchemaDS**. Then, load XML document into **OldDocumentDS**. ADO.NET will save XML schema into **DataColumn**. After this process, the term complex type becomes table and term element becomes column (see section 2.4 (2)). Finally, create a **RelationRef** (see table schema in Figure 4.9. (c)).

In the second step, the programs acquire primary key table from the user because the Hierarchical XML does not define key (see the end of section 2.2). The primary key for every table i.e.: **documentDate** for **document** table, **name** for **mortgagee** table, **loanNumber** for **mortgage**, and **name** for **bank**. Followed by save a parent table name in the level 0, in this case, is **DOCUMENT** into **ParentNodeArr** array.

In the fourth step, **document** table is in the present level (level 0). Then, programs create **document** table in **DocumentDS**. Include all columns names of the **document** table from **OldSchemaDS**, except **document_Id** (see Figure 4.1. (a)). The result **document** table schema shows at Figure 4.11. (a), until this process the contents of **document** table are **documentDate** and **comment**. After that software tool set **documentDate** as a primary key.

In the fifth step, the programs get all the child tables (the tables on level 1) of the **document** table. There are three child tables, i.e. **mortgagee**, **mortgage**, and **bank**. In this step, the child table has two possibilities. First, the child table is a leaf, for example **mortgagee**, **mortgage**, and **bank** tables. Second, the child table is not a leaf. After that, programs process every child from the most left child table until the most right child table through step 6 until step 11.

The first child, the **Mortgagee** table is processed along step 6 until step 11. Because **Mortgagee** table is a leaf then the step sixth is processed. Programs create **Mortgagee** table in **DocumentDS** and save all **Mortgagee** column names, except the **document_Id** column from **OldSchemaDS**. In addition, programs set primary key “**name**” in the creating table, **Mortgagee**. See Figure 4.1. (b) as a table source and Figure 4.11. (b) as a result table. MaxOccurs **Mortgagee** table is defined one, therefore programs save the primary key **Mortgagee** table “**name**” into **Document** table as **foreign key** in the step seventh. Furthermore, in the step eleventh, save the relationship between **Document** table and **Mortgagee** table in **RelationRef** table (see Figure 4.12. (b) the first line).

Column Name	Column Type
comment	string
documentDate	date
name	string
bankname	string

(a) Document

Column Name	Column Type
name	string
location	string
city	string
state	string
phone	string

(b) Mortgagee

Column Name	Column Type
property	string
date	date
loanAmout	decimal
term	integer
loanNumber	String
documentDate	date

(c) Mortgage

Column Name	Column Type
name	string
location	string
city	string
state	string
phone	string

(d) Bank

Figure 4.11. Flat XML Schema Result in **DocumentDS**.

The second child, the **Mortgage** table is processed along step 6 until step 11. Since **Mortgages** table is a leaf then programs process the sixth step. Programs create

Mortgagee table in **DocumentDS** and save all **Mortgage** column names, except the **document_Id** column from **OldSchemaDS**. In addition, programs set primary key “**loanNumber**” in the creating table, **Mortgage**. See Figure 4.1. (c) as a table source and Figure 4.11. (c) as a result table. MaxOccurs **Mortgagee** table is defined by eight and induplication instance, then programs save the primary key **Document** table “**documentDate**” into **Mortgage** table as **foreign key** in the step seventh. Furthermore, in the step eleventh, save the relationship between **Document** table and **Mortgage** table in **RelationRef** table (see Figure 4.12. (b) the third line).

TableName	PrimaryKey
document	documentDate
mortgagee	name
bank	name
mortgage	loanNumber

(a) Table Complex Type

Relation	ParentComplex	ParentKey	ChildComplex	ChildKey
document_mortgagee	mortgagee	name	document	name
document_bank	Bank	name	document	bankname
Document_mortgage	document	documentDate	mortgage	loanNumber

(b) Table Relational Reference

Figure 4.12. Reference Table for Transformation Hierarchical XML into Flat XML.

The third child, the **Bank** table is processed along step 6 until step 11. Because **Bank** table is a leaf then the step sixth is processed. The programs create **Bank** table in **DocumentDS** and save all **Bank** column names, except the **document_Id** column from **OldSchemaDS**. In addition, the programs set primary key in the created table, **Bank**.

Figure 4.1. (d) shows a source table and Figure 4.11. (d) shows a result table. Because maxOccurs **Bank** table is defined one, then the programs save the primary key **bank** table “**name**” into **document** table as foreign key in the step seventh. However, the column name “**name**” has already existed in **document** table so programs change the column name to become “**bankname**”. The end state of the **document** table schema is in Figure 4.11. (a). Next step, eleventh, save the relationship between **document** table and **bank** table in table **RelationRef** (see Figure 4.12. the second line).

In the twelfth step, because all child tables have already been processed then the programs delete the bottom content of **ParentNodeArr** array. In this case program will delete **document** in the array. The array is empty therefore programs stop the process.

The second major step of transformation from Hierarchical XML into Flat XML algorithms is converting Hierarchical XML Documents into Flat XML Documents algorithms. Once the Flat XML Schemas have already been created in **DocumentDS**, as the table schema, they are ready to receive the data conversion (see Figure 4.13. as a block diagram of the algorithms). The Inputs for this process are **DocumentDS** as table schemas created in the previous process and **OldDocumentDS** that Hierarchical Document is saved. The output of the process is filled data table schemas in **DocumentDS** (see Figure 4.14.). ADO.NET DataRow will be used to get the data in a DataTable.

The algorithms to convert Hierarchical XML Document into Flat XML Document are:

Step 1: Get and save a parent table name on the level 0 that will be converted into **ParentNodeArr** array.

Step 2: Check the content **ParentNodeArr** array, if the content of array is empty then terminate the process.

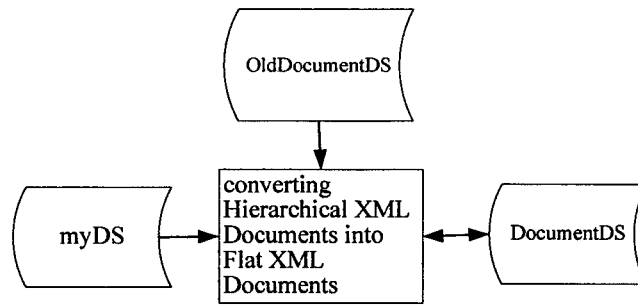


Figure 4.13. Converting Hierarchical XML Documents into Flat XML Documents Block Diagram.

Step 3: Read parent table name in **ParentNodeArr** array as a present level. Repeat

Step 4 to Step 10 until all parent table data have already been read.

Step 4: Read a row data from the parent table in **OldDocumentDS**. After that, get the **primary key** from the parent table in **DocumentDS**. The **primary key** is used to validate the row data from **OldDocumentDS**. If the data still do not exist then write the data into the table in **DocumentDS**, except **ChildColumns** and **ParentColumn**.

Step 5: Check all the child tables of the parent table on the present level from left to right because the child table has two possibilities state, either as a leaf table or not as a leaf table. Therefore, this step should check the status of child table.

Step 6: If the child table is not a leaf table, then save the name of child table into **ParentNodeArr** array. After all siblings are processed, the table just saved into **ParentNodeArr** array will become parent table for next loop.

documentDate	Comment	name	bankname
2004-07-11	Good	Hans Schmidt	Niaga
2004-07-14	Good	Hans Schmidt	XML Bank
2005-03-02	Good risk	James Blandings	XML Bank

(a) Document Table

name	location	city	state	phone
Hans Schmidt	123 Hallgarten	Berlin		870.220.5678
James Blandings	1234 299th St	New York	NY	888.555.1234

(b) Mortgage Table

Property	date	loanAmount	term	loanNumber	documentDate
Bungalow	2004-07-12	50000	12	12 3122 34	2004-07-14
Vila	2004-07-12	55000	12	11 2233 44	2004-07-11
The Hackett Place	2005-03-01	80000	15	66 7777 88	2005-03-02
123 Acorn Drive	2005-03-01	90000	15	11 8888 22	2005-03-02
House	2004-12-25	95000	24	11 1222 33	2004-07-11
99 West Pocusset St	2005-03-02	100000	30	33 4444 11	2005-03-02
19 Johnson Place	2005-03-02	110000	30	55 3333 88	2005-03-02
345 Notingham Court	2005-03-02	120000	30	22 6666 99	2005-03-02

(c) Mortgage Table

name	location	City	state	phone
Niaga	56 Sweet Street	Berlin		811.110.1234
XML Bank	12 Schema Place	New York	NY	888.555.8888

(d) Bank Table

Figure 4.14. Flat Mortgage XML Document.

Step 7: If the child table is a leaf table, then read a row data from the child table in **OldDocumentDS**. After that, get the **primary key** for the table in **DocumentDS**. The **primary key** is used to validate the row data from

OldDocumentDS. If the data still does not exist then write the data into child table in **DocumentDS**, except **ParentColumns** or **ChildColumns**. In addition, use a **DataRelation** object to know the **relation name** between the parent table and the child table. The **relation name** is used to find the column mapping in **RelationRef** table. Furthermore, this step must check the maxOccurs child table.

Step 8: If the maxOccurs child table defines one then save the **primary key child table** into **parent table**.

Step 9: If the maxOccurs child table defines many then save the **primary key parent table** into **child table**.

Step 10: Delete the bottom content of **ParentNodeArr** array and go to step 2.

The researcher applies the transforming from Hierarchical XML into Flat XML algorithms to Supplier-Part-A XML Schemas which the tree diagram shows in Figure 4.6. (a). The result is shown in Figure 4.15. The researcher also applies algorithms to Supplier-Part-C XML Schemas which the tree diagram shows in Figure 4.6. (c). The result is shown in Figure 4.16. The different while creating table schema is in **Step 4** because **Supplier_Part** table has another column **No_of_Item** except **ChildColumns** and **ParentColumns** then the program create a table of **Supplier_Part**.

4.5 The Reverse Engineering from XML Schemas into NIAM Conceptual Schemas Algorithms

A reverse engineering block diagram is shown in Figure 4.17. The inputs to reverse engineering process are the well-formatted XML Schemas and the well-formatted XML Documents and also the XML Documents have already been validated by XML Schemas (see introduced section 2.2). The **CSDP** procedure is used for the reverse engineering process. The reverse engineering process employs meta tables (see

section 4.2 and Figure 4.9., as the output of the process see Figure 4.18.).

Snumber	Sname	Status Num	City Name
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	10	Paris
S4	Clark	20	London
S5	Adams	30	Athens

(a) Supplier Table

Pnumber	Pname	ColorName	Loc Num	No of Item	Snumber
P1	Nut	Red	London	300	S1
P2	Bolt	Green	Paris	200	S1
P3	Screw	Blue	Rome	400	S1
P4	Screw	Red	London	200	S1
P5	Cam	Blue	Paris	100	S1
P6	Cog	Red	London	100	S1
P1	Nut	Red	London	300	S2
P2	Bolt	Green	Paris	400	S2
P2	Bolt	Green	Paris	200	S3
P2	Bolt	Green	Paris	200	S4
P4	Screw	Red	London	300	S4
P5	Cam	Blue	Paris	400	S4

(b) Part Table

Figure 4.15. Flat Supplier-Part XML Document in ADO.NET for Figure 4.6. (a).

The essential coding in the meta table are **Uniqueness** code and **MinCardinality** code in **Role Table** because these codes importantly play in the **forward engineering**. There are tree codes that will be used for **Uniqueness** code. “0” is to show the role in the fact type is **not unique** and “1” is to show the role in the fact type is **single uniqueness constraints**. The last is “2” to show that the role is part of the **compound**

uniqueness constraints, with the other role make uniqueness fact type. For example see Figure 4.18. and Figure 4.19., PredicateCode “P1”. “R1” has **Uniqueness** code “1” and “R2” has **Uniqueness** code “0”. Another example, see PredicateCode “P9”, “R17” and “R18” having **Uniqueness** code “2”. It means compound uniqueness constraints.

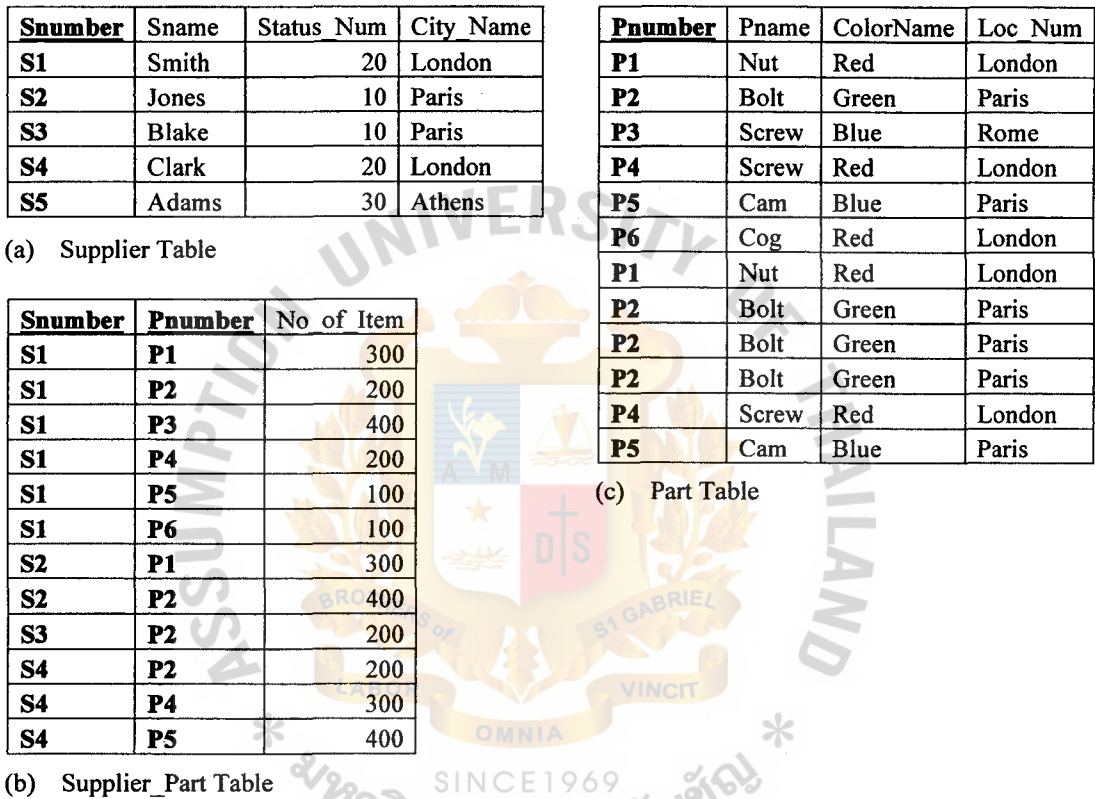


Figure 4.16. Flat Supplier-Part XML Document in ADO.NET for Figure 4.6. (c).

The other coding is **MinCardinality** code. There are two codes. “1” is used to indicate **mandatory** and “0” is used to indicate **optional**. For example, see Figure 4.18. and Figure 4.19., PredicateCode “P1”, “R1” is **mandatory** then set “1” in **MinCardinality**. Moreover, see PredicateCode “P12” **MinCardinality** code for “R24” is “0” as well “R25”.

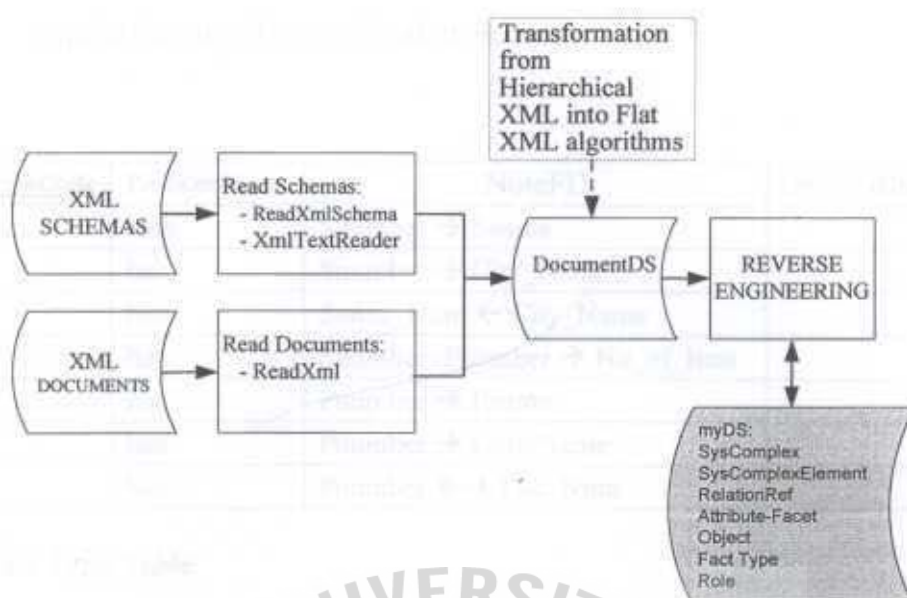


Figure 4.17. Reverse Engineering Block Diagram.

The proposed reverse engineering process will be started by checking the **uniqueness constraints**, **CSDP step 4**, followed by **checking key length**, in order to decide split fact type into an elementary fact type (see the end of section 2.3 (2)).

The algorithms to reverse engineering from XML Schemas into NIAM conceptual schemas are:

Step 1: Read XML Schema and XML Document into **DocumentDS**.

Step 2: Check the existing keys and references definition in XML Schema. If they exist then go **Step 4**. On the other hand, if they do not exist then the programs must discover the candidate key for every complex type. The discovered key must be a minimal combination of columns where no duplicates are allowed. To create the candidate keys, the programs use XML document as instances. They check the possibility of the candidate keys from the first element to the end element complex type. If with one element the uniqueness still does not

acquire then use the combination elements.

<u>PredicateCode</u>	PredicateName	NoteFD	ComplexGoal
P1	Has	Snumber \rightarrow Sname	
P3	has	Snumber \rightarrow City_Name	
P6	has	Status_Num \leftarrow City_Name	
P9	has	Snumber+Pnumber \rightarrow No_of_Item	
P10	has	Pnumber \rightarrow Pname	
P11	has	Pnumber \rightarrow ColorName	
P12	has	Pnumber \leftrightarrow Loc_Num	

(a) Fact Type Table

RoleCode	<u>PredicateCode</u>	<u>LabelType</u>	Uniqueness	MinCardinality
R1	P1	Snumber	1	1
R2	P1	Sname	0	0
R5	P3	Snumber	1	1
R6	P3	City_Name	0	0
R11	P6	Status_Num	0	0
R12	P6	City_Name	1	1
R17	P9	Snumber	2	0
R18	P9	Pnumber	2	0
R19	P9	No_of_Item	0	0
R20	P10	Pnumber	1	1
R21	P10	Pname	0	0
R22	P11	Pnumber	1	0
R23	P11	ColorName	0	0
R24	P12	Pnumber	2	0
R25	P12	Loc_Num	2	0

(b) Role Table

Figure 4.18. End State Meta Tables (Part of Appendix D).

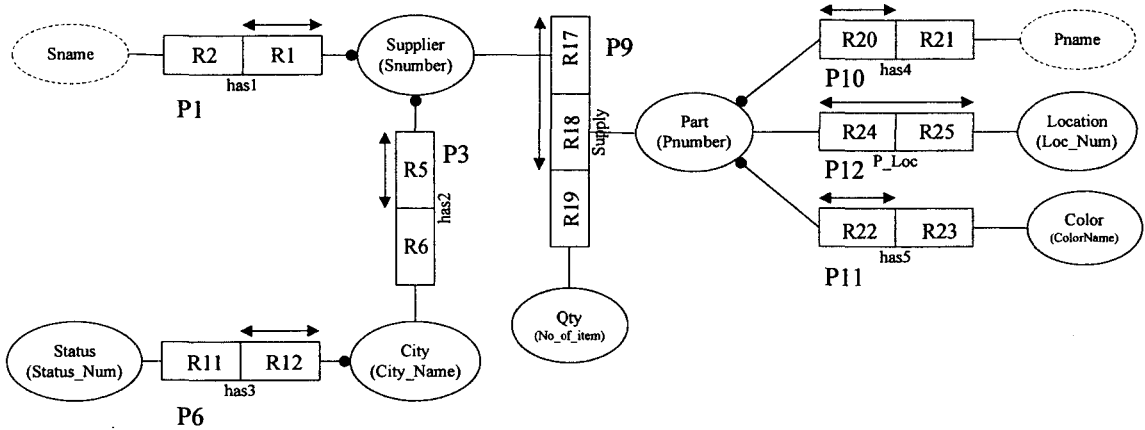


Figure 4.19. Supplier Part Conceptual Schema Diagram.

- Step 3:** Request the user to decide the key from the candidate key for every complex type.
- Step 4:** Check the key length for every complex type to slit the complex type, in order to create elementary fact types. For this purpose, the functional dependency will be employed among elements, between elements key and element non-key, and between element part of keys and element non-key.
- Step 5:** Check the created elementary fact type with data from XML Document. If functional dependency data is right then save it in the meta tables in **myDS**.
- Step 6:** Request the user to check, modify, and complete them. If users delete an elementary fact type, then the programs delete it in the **Fact Type** table. After that they delete all related rows in the **Role** table. If users change the mandatory of elementary fact type, then the programs save the modification row in the **Role** table. However, if users change the uniqueness of elementary fact type, then the programs save the modification row not only in the **Role** table but also save the modification row in the **Fact Type** table.

To make clear understanding, the researcher demonstrates the algorithms to Supplier-Part XML Schema and Supplier-Part XML Document (see Figure A.3. and Figure A.4. respectively). In the first step, the programs load XML Schema and XML Document into **DocumentDS**.

In the second step, the programs check keys and references definition in XML Schema. The XML Schema has already defined key for every complex type. For instance, **Snumber** is a key for **Supplier** and **Pnumber** is a key for **Part** complex type. Moreover the compound key, **Snumber** and **Pnumber**, is declared for **Supplier_Part** complex type. In case the XML Schemas do not define the key, the program must discover the candidate key. By using DataView facilities to sort the table by the first column until the key find then the programs check the uniqueness column in the table.

In the third step, request the user to decide the key from the candidate key for every complex type.

In the fourth, check the key length for every table. For example, Supplier table has four columns (Snumber, Sname, Status_Num, and City_Name) and the key length of **Supplier** table is one, **Snumber**. Therefore, the table is splittable. The possibilities functional dependencies are $Snumber \rightarrow Sname$, $Snumber \rightarrow Status_Num$, $Snumber \rightarrow City_Name$. In addition, it is also possible to have functional dependency among columns non key. Therefore, the checking will do to $Sname \rightarrow Status_Num$, $Sname \rightarrow City_Name$, $Sname \rightarrow Snumber$, $Status_Num \rightarrow City_Name$, $Status_Num \rightarrow Snumber$, $Status_Num \rightarrow Sname$, $City_Name \rightarrow Snumber$, $City_Name \rightarrow Sname$, and $City_Name \rightarrow Status_Num$. Actually functional dependency to realize the first step CSDP, transform familiar information examples into elementary facts, and apply quality checks. For example, $Snumber \rightarrow Sname$, it is elementary fact the supplier with supplier number "S1" has a name "Smith" (see Section 2.3 (2)).

In the step fifth, check the created elementary fact type with data from XML Document, the task to implement the second CSDP. If the relationship between entity types is functional dependency then save it in the meta tables (see Appendix E for the whole the instances meta tables).

Finally, in the step sixth, request the user to check, modify, and complete every elementary fact types because not all of functional dependencies are semantically correct, for instance see Table 4.2. For this task, user must have knowledge NIAM conceptual framework and understand the UoD (see Figure 4.19.). The user can delete the functional dependencies that are not semantically correct. “V” is used to mark the correct functional dependency. Moreover, the user also can change the relationship between columns, for example, the application tool discovers relationship between Snumber and Sname is one-to-one, $Snumber \leftrightarrow Sname$. The user can change to one-to-many, $Snumber \rightarrow Sname$. The changed meta tables are shown in Table 4.2. The final meta tables are shown in Figure 4.18.

4.6 The Forward Engineering from NIAM Conceptual Schemas into XML Schemas Algorithms

This section discusses the essential idea of the forward engineering process, see Figure 4.20. as an illustration of forward engineering block diagram. The forward engineering from NIAM conceptual schemas into XML Schemas algorithms will be separated into four major steps, i.e. transforming the NIAM conceptual schema in the **Meta Tables** into the relation schemas, converting the XML Documents in **DocumentDS** to the relations schemas, writing XML Schemas, and writing XML Documents.

Table 4.2. Correctness Suppliers and Parts Functional Dependency.

Table Name	Note FD	Correct	Correction FD
Suppliers	Snumber \leftrightarrow Sname	V	Snumber \rightarrow Sname
	Snumber \rightarrow Status_Num		
	Snumber \rightarrow City_Name	V	
	Sname \rightarrow Status_Num		
	Sname \rightarrow City_Name		
	Status_Num \leftrightarrow City_Name	V	Status_Num \leftarrow City_Name
Suppliers-Parts	Snumber+Pnumber \rightarrow No_of_Item	V	
Parts	Pnumber \rightarrow Pname	V	
	Pnumber \rightarrow ColorName	V	
	Pnumber \rightarrow Loc_Num	V	Pnumber \leftrightarrow Loc_Num

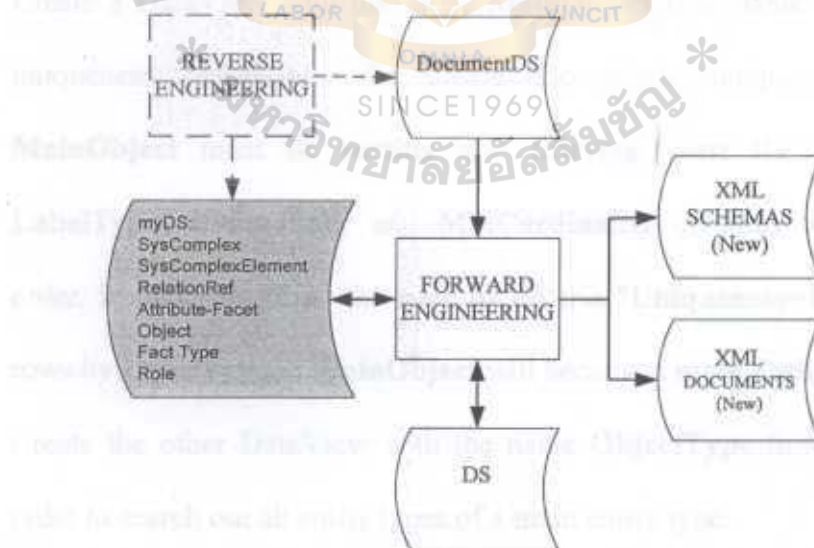


Figure 4.20. Forward Engineering Block Diagram.

The first major step of forward engineering from NIAM conceptual schemas into XML Schemas algorithms is transforming the NIAM conceptual schema from the **Meta Tables** into the relation schemas. The created **Meta Tables** in **myDS** and **XML Schema** and **XML Document** in **DocumentDS** from the reverse engineering process are the input to the forward engineering process. **Rmap** procedure is used for the process transformation. The transformed XML Schema and the converted XML Document are temporarily saved in **DS** before they are written into XML Schema and XML Document.

Conceptually the Rmap procedure works by grouping fact type into table schema (see Section 2.3 (3) and Figure 4.21.). Figure 4.21. gives an illustration how Rmap procedure is adopted in this research. Moreover, Figure 4.22. shows how to map the NIAM Schemas in Meta Tables into Relational Schemas.

The transforming NIAM conceptual schema in the **Meta Tables** into the relation schemas algorithms are:

- Step 1:** Create a DataView with the name **MainObject** from **Role** Table for single uniqueness constraints. To create the single uniqueness constraints, **MainObject** must be **sorting** and **filtering**. Sort the **Role** Table by **LabelType**, **Uniqueness**, and **MinCardinality** columns with **descending** order. In addition, **filter** the table by criteria "**Uniqueness=1**". The group of rows by **LabelType** in **MainObject** will become a **main entity type**.
- Step 2:** Create the other DataView with the name **ObjectType** from **Role** Table in order to search out all entity types of a main entity type.
- Step 3:** Create a table for every main entity type. Every row in **ObjectType**, without duplication **LabelType**, will become a column in the created table. For this purpose, check the mandatory, if it is "0" then the row directly become a

column. Otherwise check whether object type has another functional role. Based on **LabelType** in **ObjectType** find column name and type in **SysComplexElement** Table. Then save the created table in **FactType** Table, column **ComplexGoal**. Finally create a primay key for every **Uniqueness="1"**.



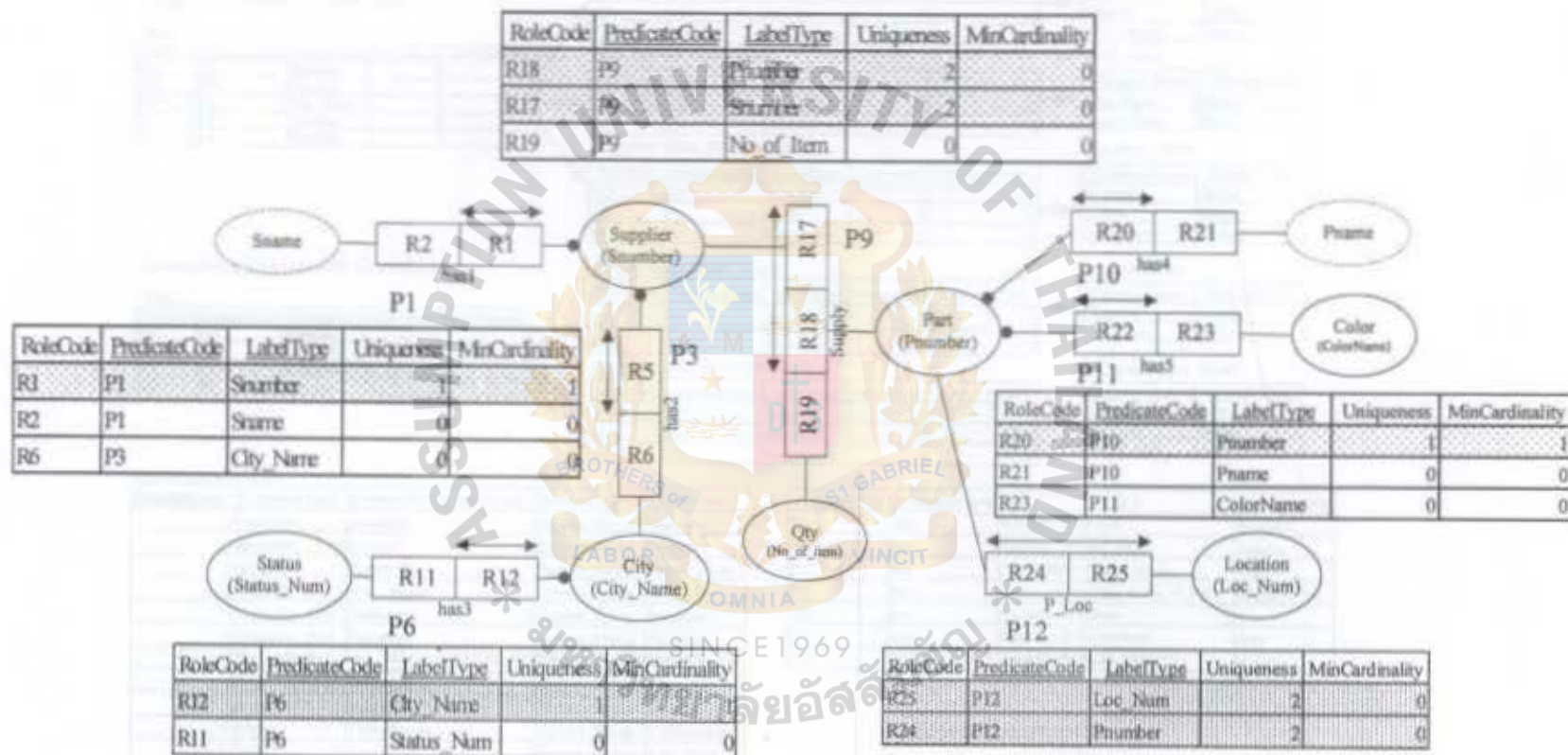


Figure 4.21. Grouping the Elementary Fact Types in Meta Tables by the Main Entity Type.

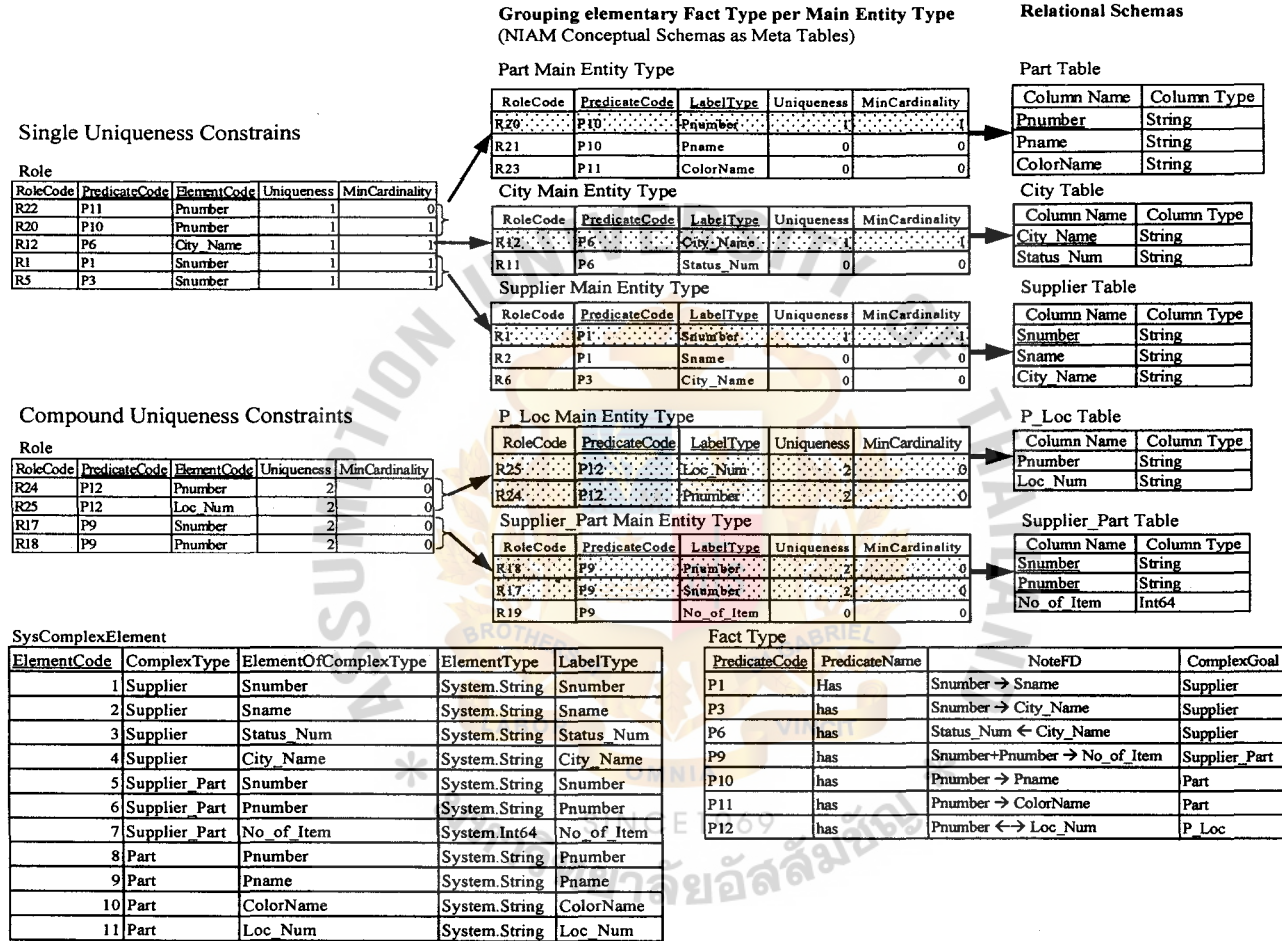


Figure 4.22. Mapping NIAM Schemas in Meta Tables into Relational Schemas.

Step 4: Create a DataView with the name **MainObject** from **Role** Table for compound uniqueness constraints. To create the compound uniqueness constraints, **MainObject** must be **sorting** and **filtering**. **Sort** the **Role** Table by **PredicateCode** and **LabelType** columns with **ascending** order. In addition, **filter** the table by criteria "**Uniqueness=2**". The group of rows by **PredicateCode** in **MainObject** will become a **main entity type**.

Step 5: Create the other DataView with the name **ObjectType** from **Role** Table for every main entity type with compound uniqueness constraints in **MainObject**.

Step 6: Create a table for every main entity type. Every row, without duplication **LabelType**, in **ObjectType** will become a column in the created table. Based on **LabelType** in **ObjectType**, find table name, column name, and column type in **SysComplexElement** Table. After that, save the created table name in **FactType** Table, column **ComplexGoal**. Finally create compound primary key based on the row with **Uniqueness="2"**.

The researcher demonstrates the algorithms to meta tables in **myDS**. In the first step, since each relational table stores one or more elementary fact types, therefore, the software tool must know what elementary fact types have functional roles that attached to the same object type, and the main entity type. For this purpose, the Role Table is sorted and filtered. For example, to find the main entity types with single uniqueness constraint, sort the Role Table by **LabelType**, **Uniqueness**, and **MinCardinality** columns with **descending** order. In addition, filter the table by criteria "**Uniqueness=1**". Figure 4.23. (a) shows the result of the process. There are tree groups, i.e. group with **LabelType** "**Pnumber**", "**City_Name**", and "**Snumber**". Moreover, the Figure shows the first group has two fact types with **PredicateCode** "**P11**" and "**P10**". However, the second group only has one fact type with

PredicateCode “P6”. The third group also has two fact types with **PredicateCode** “P1” and “P3”. Furthermore, the table showing the **LabelType** will become a primary key, indicated by **Uniqueness** “1”.

RoleCode	PredicateCode	LabelType	Uniqueness	MinCardinality	
R22	P11	Pnumber	1	1	First Group
R20	P10	Pnumber	1	1	
R12	P6	City_Name	1	1	Second Group
R1	P1	Snumber	1	1	Second Group
R5	P3	Snumber	1	1	

(a) Main Entity Types with Single Uniqueness Constraints

RoleCode	PredicateCode	LabelType	Uniqueness	MinCardinality
R20	P10	Pnumber	1	1
R22	P11	Pnumber	1	0
R21	P10	Pname	0	0
R23	P11	ColorName	0	0

(b) Main Entity Type for the First Group (LabelType “Pnumber”)

Figure 4.23. Grouping Fact Type for the Single Uniqueness Constraints.

In the second step, to discover all elementary fact types, for instance the first group with **LabelType** “**Pnumber**”, filter the Role Table with the **PredicateCode** in the first group. See Figure 4.23. (b) as a result of the process.

The third step, the table is used as base to transform Meta Tables into relational schemas. All MinCardinality are “0” except the first line. The first line, with Uniqueness and MinCardinality “1”, is the uniqueness constraint. The other lines directly become a column of table.

In the step fourth, to find the main entity types with compound uniqueness

constraint, sort the **Role** Table by **PredicateCode** and **LabelType** columns with **ascending** order. In addition, filter the table by criteria “**Uniqueness=2**”. Figure 4.24. (a) shows the result of the process. There are two groups, i.e. group with **PredicateCode** “**P12**” and “**P9**”. Every row with **Uniqueness=2** will become part of compound uniqueness constraints key. The table is used as base to transform Meta Tables into relational schemas in the step sixth. At the end of the algorithms, relational schema in the fifth normal form in the most top right Figure 4.22 is created.

RoleCode	<u>PredicateCode</u>	<u>LabelType</u>	Uniqueness	MinCardinality	
R25	P12	Loc_Num	2	0	First Group
R24	P12	Pnumber	2	0	
R18	P9	Pnumber	2	0	Second Group
R17	P9	Snumber	2	0	

(a) Main Entity Types with Compound Uniqueness Constraints

RoleCode	<u>PredicateCode</u>	<u>LabelType</u>	Uniqueness	MinCardinality
R18	P9	Pnumber	2	0
R17	P9	Snumber	2	0
R19	P9	No_of_Item	0	0

(b) Main Entity Type for the Second Group (PredicateCode “P9”)

Figure 4.24. Grouping Fact Type for the Compound Uniqueness Constraints.

As mentioned in section 2.7, the researcher adopts the transformation of NIAM conceptual schema into XML Schema with modifying the one-to-one relationship transformation proposed by Chankuang and Suphamit. Figure 4.25. shows how to map one-to-one relationship from conceptual schema diagram to relational schema. It is discussed in section 2.3 (3) Figure 2.16. In this discussion, the researcher wants to

shows how map one-to-one relationship from meta table to relational schema with considering default procedure for mapping one-to-one fact type.

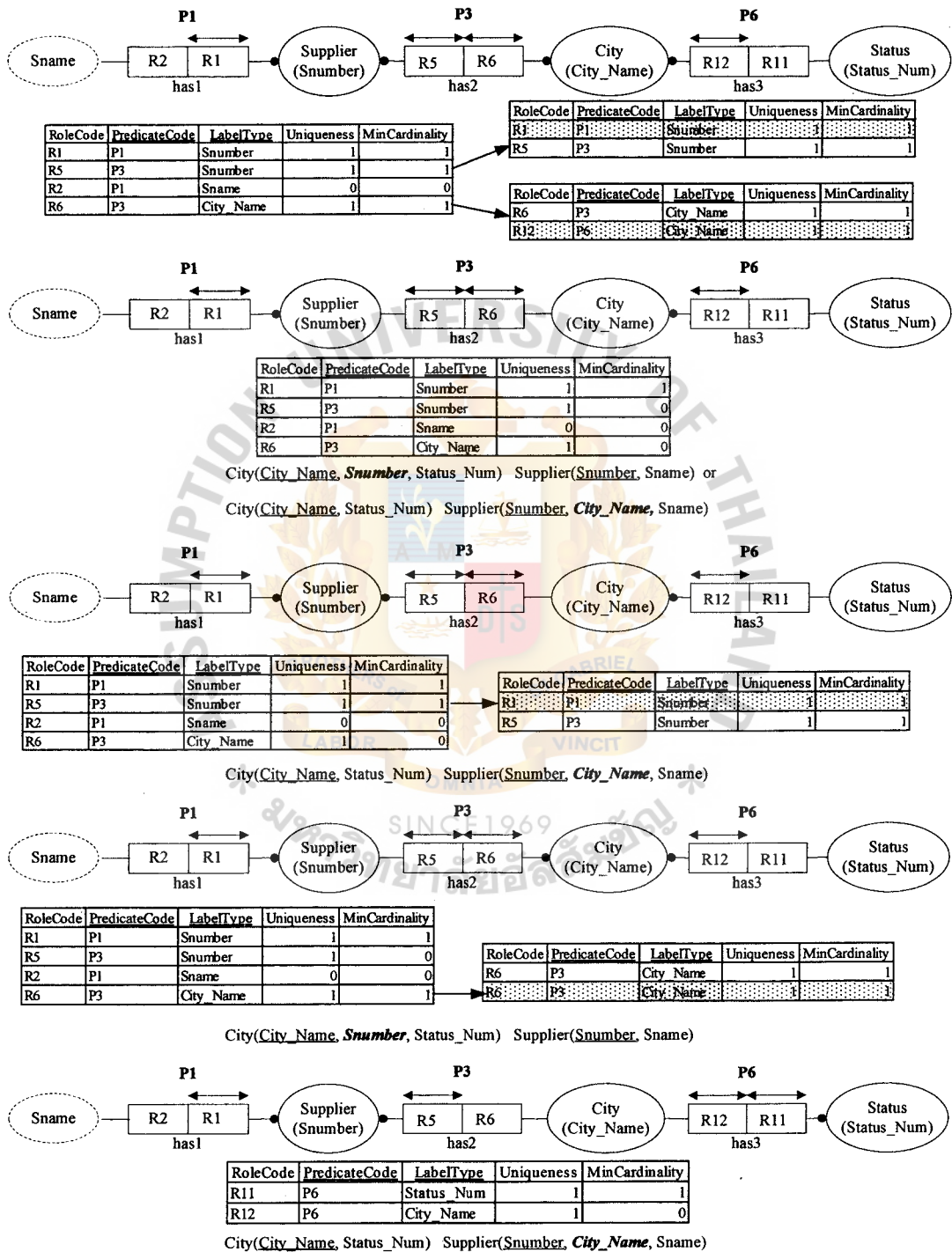


Figure 4.25. Mapping One-to-One Relationship for Supplier and City Meta Table into Relational Schema.

While MinCardinality is “1” then the software tool must check whether the object type (LabelType) in the one-to-one predicate has another functional role. If it has another functional role, then group on its side.

The second major step of forward engineering from NIAM conceptual schemas into XML Schemas algorithms is converting the XML Documents in **DocumentDS** to the relations schemas (see Figure 4.26.). Once the relational schemas created from the previous major algorithms, they are ready to receive the data conversion. The XML Documents that have already loaded into **DocumentDS** from reverse engineering process as the input of the conversion process, whereas, the output of the process is the filled tables in **DS** (see Figure 4.27.).

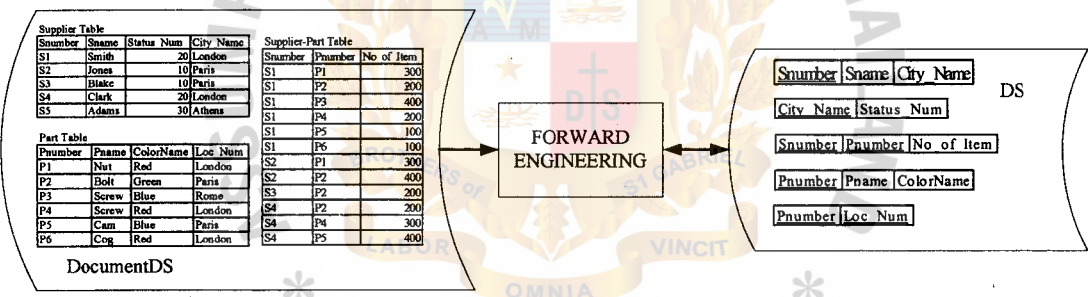


Figure 4.26. Conversion Data to New Relation Schemas.

Converting the XML Documents in **DocumentDS** to the created relations schemas algorithms are:

- Step 1:** Obtain the table name, the table schemas, and the table primary key that will be filled, and the table name of the source data.
- Step 2:** Validate the rows source data by the primary key in the filled table. While they are still not existing in the filled table then save the columns source data in the

corresponding columns of the filled table.

Snumber	Sname	City Name
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris
S4	Clark	London
S5	Adams	Athens

(a) Supplier Table

City Name	Status Num
London	20
Paris	10
Athens	30

(b) City Table

Pnumber	Pname	ColorName
P1	Nut	Red
P2	Bolt	Green
P3	Screw	Blue
P4	Screw	Red
P5	Cam	Blue
P6	Cog	Red

(c) Part Table

Snumber	Pnumber	No of Item
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

(d) Supplier_Part Table

Pnumber	Loc Num
P1	London
P2	Paris
P3	Rome
P4	London
P5	Paris
P6	London

(e) P_Loc Table

Figure 4.27. Filled Normalize the Fifth Normal Form Supplier Part Tables.

The third major steps of forward engineering from NIAM conceptual schemas into XML Schemas algorithms is writing XML Schema algorithm. Figure 4.28. shows the block diagram for writing XML Schema from **DS**. The input to this process is **DS** and **myDS** created from two previous algorithms. The output of the process is XML Schemas (see Figure B.3. and Figure B.4. for the created Supplier-PartNew XML Schema and the created Supplier-PartNew XML Document respectively).

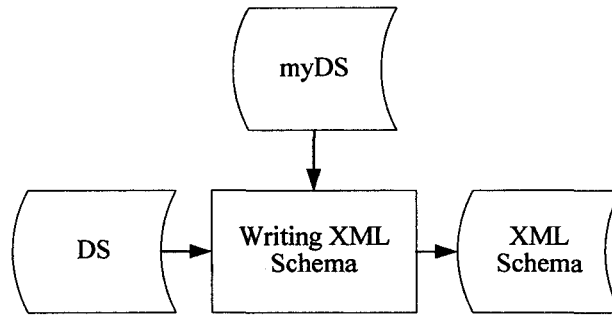


Figure 4.28. Writing Flat XML Schema Block Diagram.

The algorithm for writing XML Schemas:

Step 1: Define XML declaration.

Step 2: Define `<xsd:schema>` element, which is the element root of all XSD schema documents. In addition, on the `<xsd:schema>` element a namespace declaration for the XSD schema namespace is also declared. By default in software tool the name of element is “NIAM”

Step 3: Define complex type for every table in DataTable. The table name becomes the complex type name. Moreover, very column in DataColumn for each table in DataTable becomes element name of the complex type. In addition, very data type column in DataColumn for each table in DataTable will become element type of the complex type.

Step 4: Define the end tag element root, i.e.: `</xsd:choice>` and `</xsd:complexType>`.

Step 5: Define the key for every complex type with the following sub step, first define the name attribute of key element with the complex type name and additional suffix Key. After that, define selector element with assigning xpath attribute with the complex type name. Finally, define field element with assigning xpath attribute with the attribute name of the elements, which are assigned as

primary key. In case a complex type has composite key, it has just defined field element more than one.

Step 6: Define references for every complex type with the following sub step. First, define the name attribute of keyref element with the relation name (as source see Figure 4.29. (a), Relation column), followed by defining refer attribute with the name attribute of the key element that will be referred (as source see Figure 4.29. (a), ParentComplex column with additional suffix Key). After that, define selector element with assigning xpath attribute with the name of child complex (as source see Figure 4.29. (a), ChildComplex column). Finally, define field element with assigning xpath attribute with the name of child key (as source see Figure 4.29. (a), ChildKey column).

Step 7: Define the end tag schema, i.e.: `</xsd:element>` and `</xsd:schema>`.

The researcher demonstrates how algorithm work, it is discussed by example with difference approach conducted by Chankuang and Suphamit (see section 2.6). They use conceptual schema diagram while the researcher uses relational table created from meta tables, but it is similar (see Figure B.3. for full listing of Supplier_PartNew XML Schema). The researcher will separate the lines of listing into group of lines, as they are discussed in the step.

The first step, the software tool writes XML document declaration, such as: `<?xml version="1.0" encoding="utf-8" ?>`, in the first line of XML Schema.

The second step, the software tool writes the `<xsd:schema>` element looks like below:

```
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
3   <xsd:element name="NIAM" msdata:IsDataSet="true"
    msdata:EnforceConstraints="true">
4     <xsd:complexType>
5       <xsd:choice maxOccurs="unbounded">
```

Relation	ParentComplex	ParentKey	ChildComplex	ChildKey
Supplier_Supplier_Part	Supplier	Snumber	Supplier_Part	Snumber
Supplier_Part_Part	Part	Pnumber	Supplier_Part	Pnumber
Supplier_City	City	City_Name	Supplier	City_Name
P_Loc_Part	Part	Pnumber	P_Loc	Pnumber

(a) RelationRef Table in ADO.NET

```

73 <xsd:keyref name="Supplier_Supplier_Part" refer="SupplierKey">
74   <xsd:selector xpath="..//Supplier_Part" />
75   <xsd:field xpath="Snumber" />
76 </xsd:keyref>
77 <xsd:keyref name="Supplier_Part_Part" refer="PartKey">
78   <xsd:selector xpath="..//Supplier_Part" />
79   <xsd:field xpath="Pnumber" />
80 </xsd:keyref>
81 <xsd:keyref name="Supplier_City" refer="CityKey">
82   <xsd:selector xpath="..//Supplier" />
83   <xsd:field xpath="City_Name" />
84 </xsd:keyref>
85 <xsd:keyref name="P_Loc_Part" refer="PartKey">
86   <xsd:selector xpath="..//P_Loc" />
87   <xsd:field xpath="Pnumber" />
88 </xsd:keyref>

```

(b) KeyRef Definition for Relation Definition in XML Schema

Figure 4.29. Defining KeyRef in XML Schema.

The third step, the software tool creates a complex type schema for every table schema in **DataColumn DS** (see Figure 4.30. (a)), **Supplier** table has column Snumber, Sname, and City_Name. The columns name becomes name attribute for each element (see Figure 4.30. (b)). Furthermore, columns type become type attribute (see Figure 4.30. (b)). To write the other complex type schemas use the same way, see Figure 4.22 the top most right tables, Relational Schema 5NF. To write **Part** complex type, see the first schema table and the XML schema see Figure B.3. line 15 until line 23. Next, to write **City** complex type, see the second schema table and the XML

schema see Figure B.3. line 24 until line 31. In addition, to write **Supplier_Part** complex type, see the last schema table and the XML schema see Figure B.3. line 32 until line 40. Finally, to write **P_Loc** complex type, see the fourth schema table and the XML schema see Figure B.3. line 41 until line 48.

Column Name	Column Type
Snumber	String
Sname	String
City_Name	String

(a) Supplier Table Schema in ADO.NET

```
6 <xsd:element name="Supplier">
7   <xsd:complexType>
8     <xsd:sequence>
9       <xsd:element name="Snumber" type="xsd:string"/>
10      <xsd:element name="Sname" type="xsd:string" />
11      <xsd:element name="City_Name" type="xsd:string" />
12    </xsd:sequence>
13  </xsd:complexType>
14 </xsd:element>
```

(b) Supplier Table Schema in XML Schema

Figure 4.30. Defining Supplier Table Schema as a Supplier Complex Type.

The fifth step, define the key for every complex type. For instance see Figure 4.31. (a) the first line, the table name is Supplier and the primary key for that table is Snumber. The software tool writes XML Schema such as in Figure 4.31. (b) line 51 until 54. It writes key element with name attribute of **Supplierkey**, and after that, writes selector element with xpath attribute “**../Supplier**”. Finally, it writes field element with xpath attribute **Snumber**. To define the other key use the same way above.

The last major steps of forward engineering from NIAM conceptual schemas into XML Schemas algorithms is writing XML Document algorithms. Figure 4.32. shows the block diagram for writing XML Document from **DS**. The input to this process is **DS** created from the two first major steps of forward engineering from NIAM conceptual schemas into XML Schemas algorithms. The output of the process is XML Documents. The algorithm for writing Flat XML Document are:

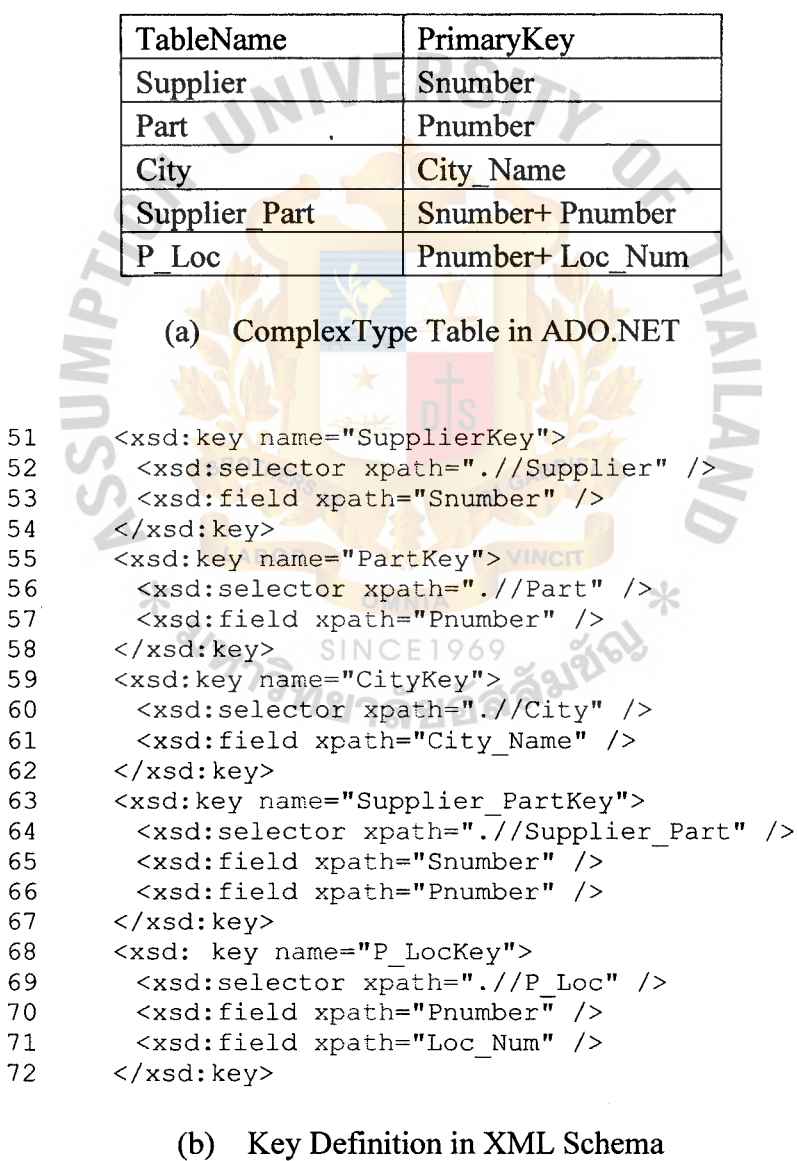


Figure 4.31. Defining Key in XML Schema.

- Step 1:** Define XML document declaration, i.e. version, encoding, and standalone.
- Step 2:** Define the start root element with the name of **DS**, followed with the name space of XMLSchema-instance and lastly the location and name of XML Schema.
- Step 3:** Define a complex type instance for every row in DataTable alongside with each complex type schema. The table name will become the start and end tag complex type name. Moreover, every column name in DataColumn for each table in DataTable will become the start and end tag element name of the complex type.
- Step 4:** Define the end root element, i.e.: <NIAM>. As a notice the name of end root element must same with start root element.

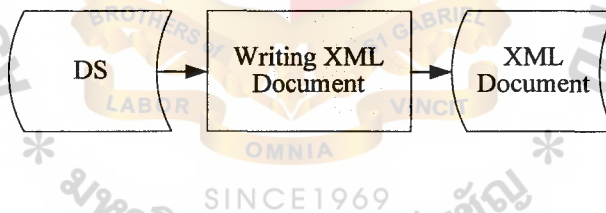


Figure 4.32. Writing Flat XML Document Block Diagram.

To show how algorithm works, the research discusses it by example. See Figure B.4. for the full listing of Supplier_PartNew XML Document. The researcher will separate the lines of listing into group of lines, as they are discussed in the step.

The first step, the software tool writes XML document declaration, such as: <?xml version="1.0" encoding="utf-8" standalone="yes"?>, in the first line of XML Document.

The second step, the software tool writes the start root element in the second line

of XML Document, such as: `<NIAM xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance xsi:noNamespaceSchemaLocation="MortgageNew.xsd">`. It gives the name of root element “NIAM”, however the value of `noNamespaceSchemaLocation` depend on the name of XML Schema processing.

The third step, the software tool creates a complex type instance for every row in DataTable alongside with each complex type schema. For example, Figure 4.33. (a) shows the instances in **Supplier** table. The table has column names, i.e. **Snumber**, **Sname**, and **City_Name**. The column name is used as the start and end tag element, as a container for a column data, for instance `<Sname>...</Sname>`. Furthermore, the table name will become the start and end tag complex type name, as a container for a row data, for example `<Supplier>...</Supplier>`. Figure 4.33. shows the definition of the instances **Supplier** table in XML Document. To define the other instances see Figure 4.27 and Figure B.4. To define instance **City**, see Figure 4.27. (b) and XML Document line 58 until line 69. Next, to define instance **Part**, see Figure 4.27. (c) and XML Document line 28 until line 57. Moreover, define instance **Supplier_Part**, see Figure 4.27. (d) and XML Document line 70 until line 129. Finally, to define instance **P_Loc**, see Figure 4.27. (e) and XML Document line 130 until line 153.

4.7 Input Design and Output Design Software Tool

This section discusses input and output design to support designing software tool. There are two kinds of design, i.e. input design and output design. The input designs are Main Menu, Analysis Sub Menu, Show Sub Menu, XDS Open File Dialog, Fransform Hierarchy to Flat XML, XML Schema Information, NIAM Conceptual Schema, Confirmation Deletetation Fact Type, and Modifying and Confirmation Fact Type Information. The output design is Display XML File.

<u>Snumber</u>	Sname	City_Name
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris
S4	Clark	London
S5	Adams	Athens

(a) Supplier Table Instance in ADO.NET

```

3      <Supplier>
4          <Snumber>S1</Snumber>
5          <Sname>Smith</Sname>
6          <City_Name>London</City_Name>
7      </Supplier>
8      <Supplier>
9          <Snumber>S2</Snumber>
10         <Sname>Jones</Sname>
11         <City_Name>Paris</City_Name>
12     </Supplier>
13     <Supplier>
14         <Snumber>S3</Snumber>
15         <Sname>Blake</Sname>
16         <City_Name>Paris</City_Name>
17     </Supplier>
18     <Supplier>
19         <Snumber>S4</Snumber>
20         <Sname>Clark</Sname>
21         <City_Name>London</City_Name>
22     </Supplier>
23     <Supplier>
24         <Snumber>S5</Snumber>
25         <Sname>Adams</Sname>
26         <City_Name>Athens</City_Name>
27     </Supplier>

```

(b) Supplier Instance from Figure B.4. Supplier_PartNew XML Document

Figure 4.33. Defining the Instances Supplier Table in XML Document.

Main menu software tool appears in Figure 4.34. The main menu content two sub menu, i.e. analysis and show, see Figure 4.35. and Figure 4.36. respectively. From sub menu analysis, users can select the XML Schemas file that will be analyzed through XSD Open File Dialog in Figure 4.37. Software tools analyze the selected XML

Schemas to know Hierarchy or Flat XML. If software tools detect the file is Hierarchy then software tools ask the user to select key for every complex type (see Figure 4.38.). To show XML Schema Information and if it is needed the user is requested to select the key for every complex type (see Figure 4.39.). Furthermore, the reverse engineering is shown in Figure 4.40, NIAM Conceptual Schema Screen. That screen also can delete or modify fact type. To confirm deleted Fact Type the user must answer the dialog that shows in Figure 4.41. Users can modify uniqueness and mandatory constraint in the screen that shows in Figure 4.42.

In this software tools, the researcher only designs output to the screen. There are several reasons. First, the output XML Schema and XML Document are not used for documentation but they will be transferred. Second, because XML Schema and XML Document are text file therefore they can be printed by every word processing.

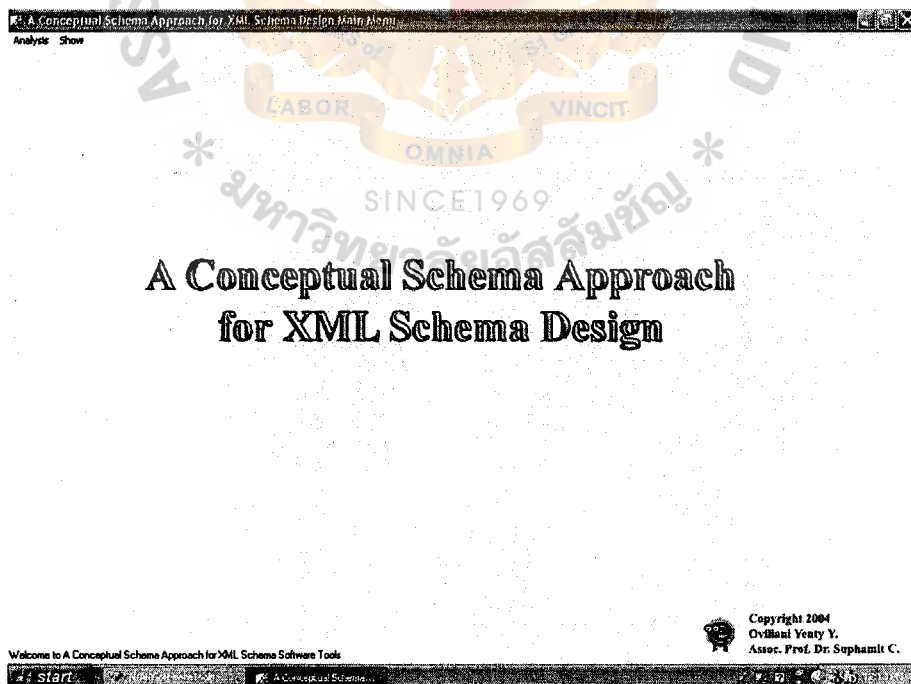


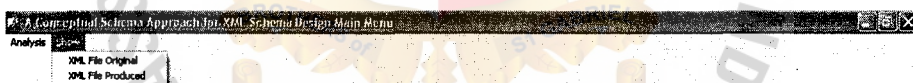
Figure 4.34. Main Menu.



A Conceptual Schema Approach for XML Schema Design



Figure 4.35. Analysis Sub Menu.



A Conceptual Schema Approach for XML Schema Design



Figure 4.36. Show Sub Menu.

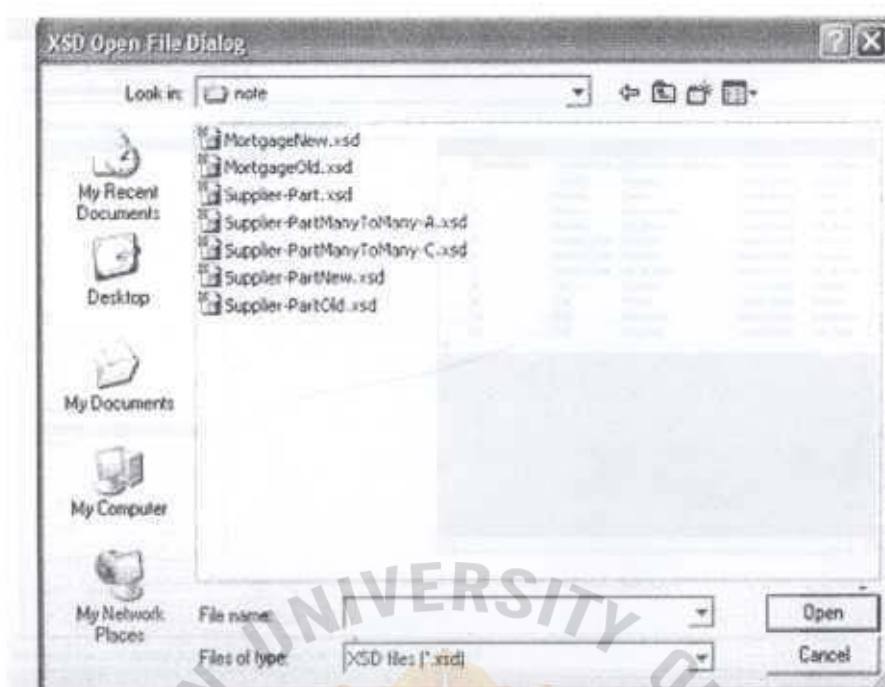


Figure 4.37. XSD Open File Dialog.



Figure 4.38. Transform the Hierarchical to the Flat XML.

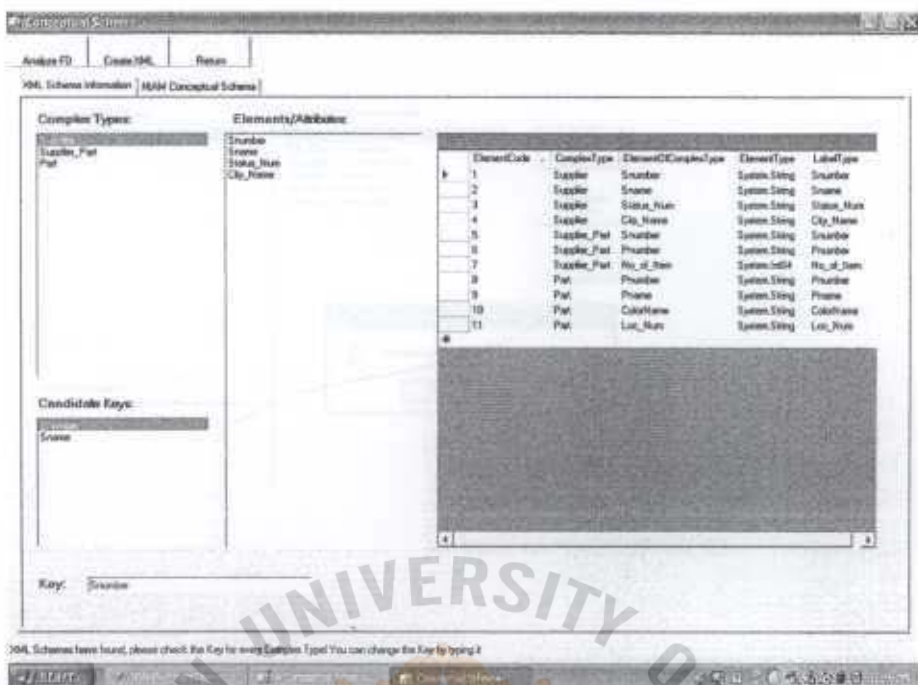


Figure 4.39. XML Schema Information.

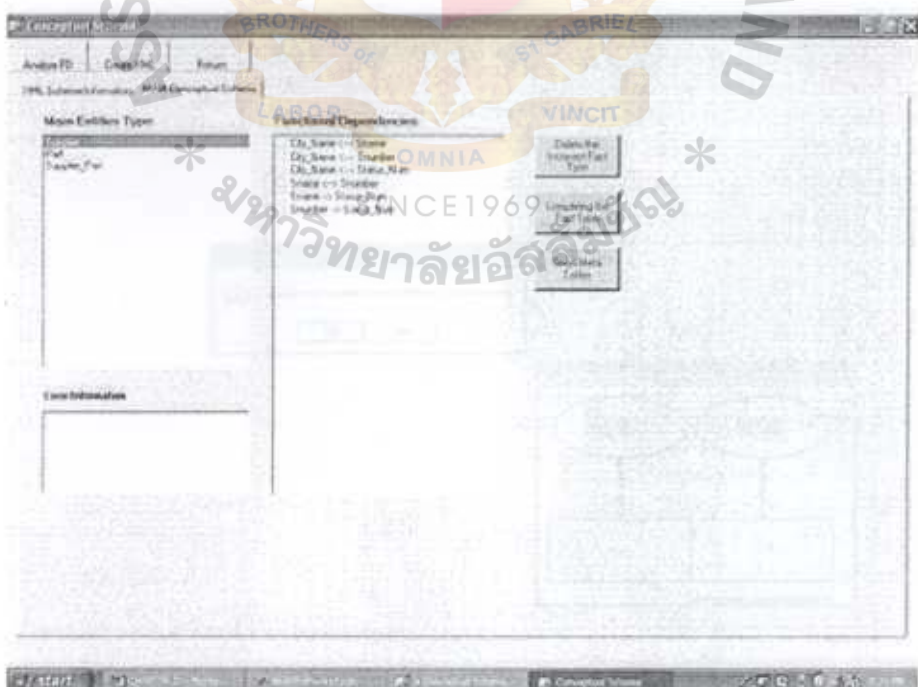


Figure 4.40. NIAM Conceptual Schema.

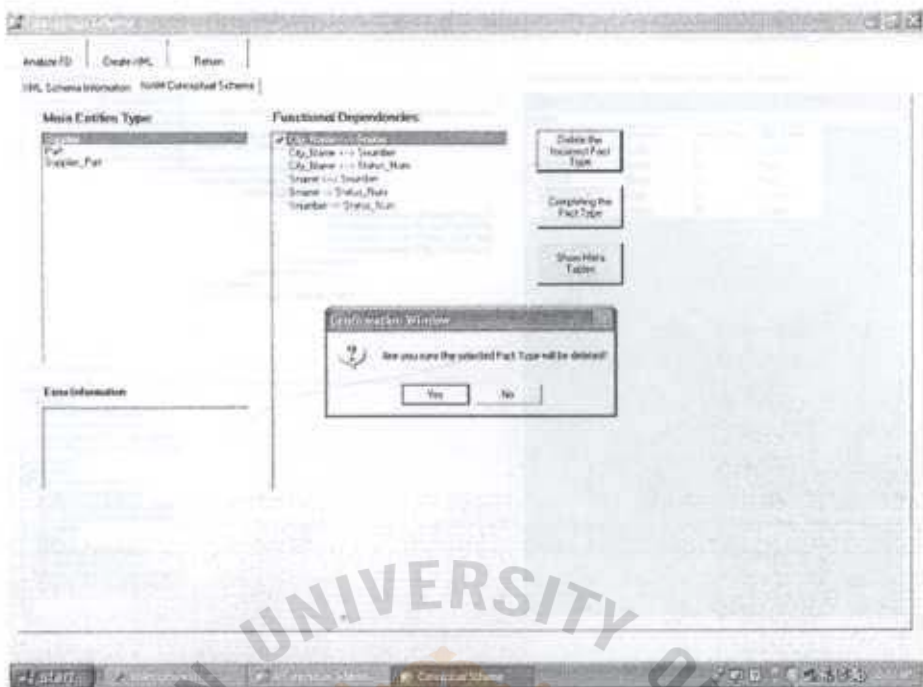


Figure 4.41. Confirmation the Deleting Fact Type.

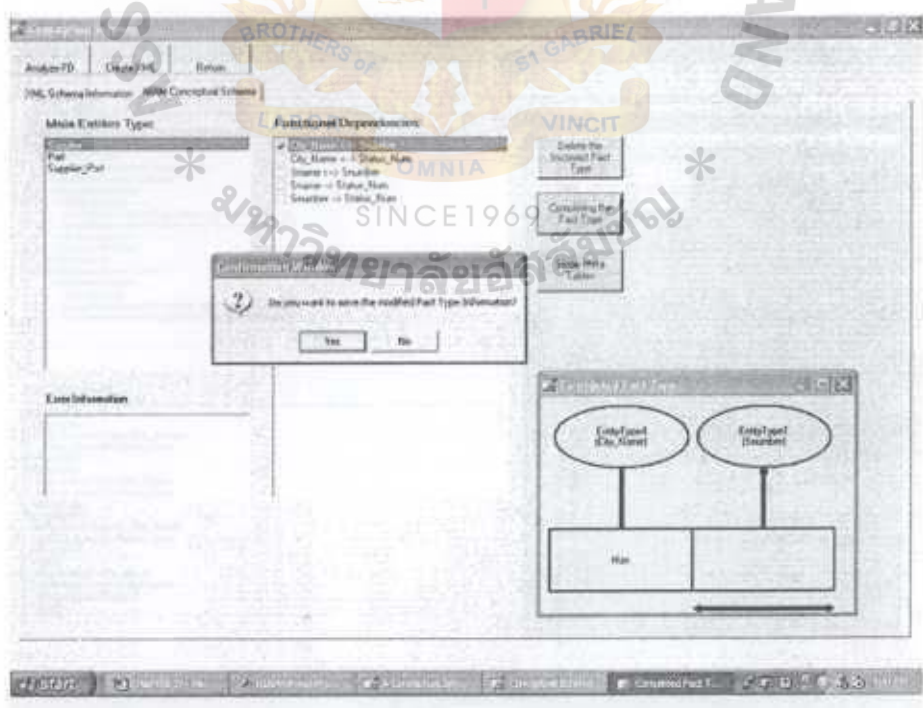


Figure 4.42. Modification and Confirmation Fact Type Information.

V. SYSTEM EVALUATION

5.1 Evaluation to XML Editors

The recent XML Editors give a basic XML syntax error to indicate the XML Schemas and XML Documents are well-formed. Several XML Editors give a feature to validate an XML Document against a XML Schema. However, no XML Editor has capability to provide information about the correctness or incorrectness of the created XML Schema nor normalizes the XML Schema.

In this study, the researcher uses XMLwriter as XML Editors to check well-formed the XML Schemas and to check well-formed and validate the XML Documents. The XMLwriter is quite good for checking, however it does not have feature to convert data from database to XML Schemas and XML Documents and also it is not good to visualize XML Schemas and XML Documents. To visualize them, the researcher uses Microsoft's Visual Studio .NET.

With limitation XML Editors, in evaluation to the created XML Schemas and XML Documents the researcher just can prove the outputs are well-formed and validated. The researcher can not prove the correctness of XML Schema. However, since the researcher follows the recommended XML Schema by W3C, it is guaranteed that the outputs are correct. Moreover, while XML Editors nor Database application is not over the feature to normalize the XML Schema, it is a good opportunity and challenge for researcher to enrich them.

5.2 Evaluation to VB.NET and ADO.NET

First problem, the researcher discusses attribute and data facet XML that can not be captured into DataColumn using ReadXmlSchema method ADO.NET in section 4.2. There are several attributes in XML can be captured in DataColumn ADO.NET but

quite a lot can not be captured. In addition, there is only one facet element in XML can be captured in DataColumn ADO.NET but quite a lot can not be captured. Therefore some information will be lost. For example, **maxOccurs** and **minOccurs** attributes can not be captured in DataColumn. However, **maxOccurs** attribute plays important role in transforming Hierarchical XML to Flat XML. Furthermore, **minOccurs** is very important in forward engineering process.

The second problem, Holzner (2004) says that not many XML processors support **schemaLocation** attribute yet, such as VB.NET but Internet Explorer and XMLwriter do. With defining XML Schemas in XML Documents while XML Documents is loaded to DataSet using **ReadXml** method and **XmlReadMode "Auto"**. VB.NET automatically loads XML Schemas. However, VB.NET does not automatically load XML Schemas. Therefore, XML Documents become text file and do not mind the defined data type in element. In addition, the researcher also can not validate XML Document against with XML Schema in Microsoft's Visual Studio .NET.

The first advantage of ADO.NET is DataTable that can be accessed by DataColumn and DataRow. Using **ReadXmlSchema** method, ADO.NET can load XML Schemas into DataColumn and using **ReadXml** method, ADO.NET can load XML Documents into DataRow. Therefore, accessing XML Schemas and XML Documents are not as a text file anymore but as a relational table. Furthermore, to access child rows, just use **GetChildRow** method. On the other hand to get parent row, just use **GetParentRow** method.

The second advantage of ADO.NET is DataView. With DataView, the researcher can **filter**, **sort**, and **search** the **contents** of **DataTable** (see section 2.4 (3)). The sorting DataView supports the researcher to check the uniqueness identifier. In

addition, filtering, sorting, and searching DataView support the researcher in reverse engineering XML Schema (split complex type element), forward engineering XML Schema (grouping element in complex element), and converting XML Documents to the new schema of XML Documents.

5.3 Evaluation to Algorithm

As mentioned in the scope of study in Chapter I about limitation and what will be implemented, the transforming from Hierarchical XML into Flat XML algorithms, the reverse engineering from XML Schemas into NIAM conceptual schemas algorithms, and the forward engineering from NIAM conceptual schemas into XML Schemas algorithms have already been implemented in software tool well. Later, the researcher tests, validates, and performs simulation to software tool.

The researcher tried two methods for writing XML Schemas from DataSet, **WriteXmlSchema** method ADO.NET and **XmlTextWriter** method. The **WriteXmlSchema** method is implemented with one statement, mention the DataSet name and the XML Schema name that will be written to, for example, `NewDs.WriteXmlSchema("MortgageNew.xsd")`. It will write XML Schemas automatically, for instance see the output in Listing B.1. and the visualization of XML Schema is shown in Figure 5.1. The **XmlTextWriter** method, however, the researcher must mention into several statements, such as `WriteStartDocument`, `WriteStartElement`, `WriteAttributeString`, `WriteElementString`, `WriteEndElement`, `WriteEndDocument`, `ElementKey`, etc. With **XmlTextWriter** method the researcher can write data facets that can not do with **WriteXmlSchema** ADO.NET. The visualization of XML Schema created with this method is shown in Figure 5.2. The figure shows as the hierarchical schema, actually should be the flat schema. It causes that researcher can not write

msdata:IsDataSet="true". No attributes in XML Schema define it. This problem is another reason in the study that the researcher concerns only on stored fact type.

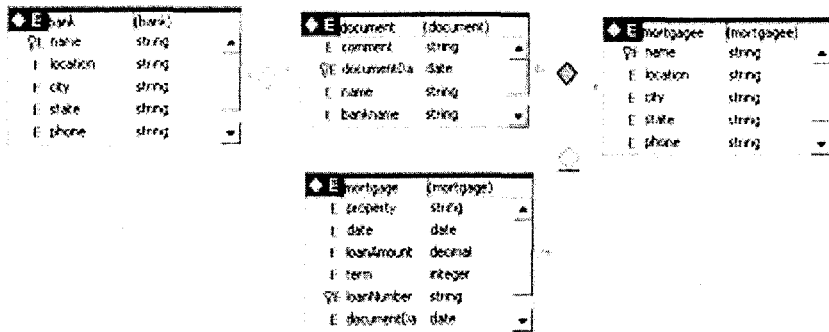


Figure 5.1. Visualization the Right MortgageNew XML Schema

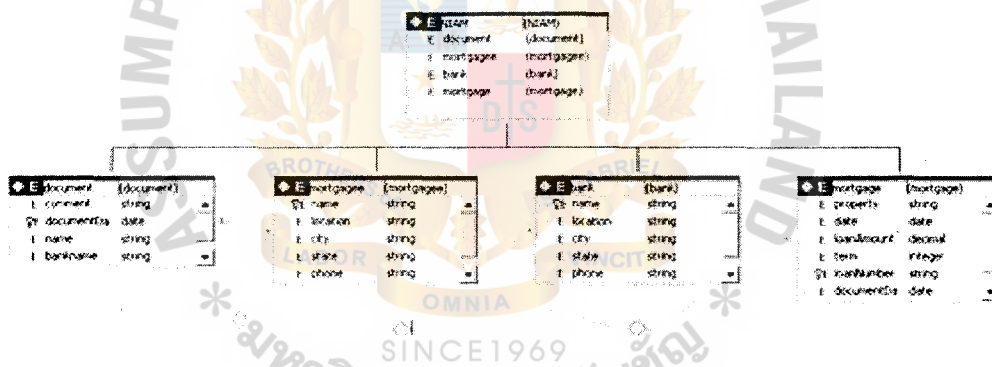


Figure 5.2. Visualization the Wrong MortgageNew XML Schema

5.4 Evaluation to the Created XML Schemas and XML Documents

First, the researcher checks the created XML Schemas and XML Documents in well-formed and validated using XML Writer. Then, the researcher checks the created MortgageNew Xml Schema in well-formed. The schema is well-formed, see Figure C.1. Moreover, the researcher checks the created MortgageNew XML Document in well-formed and validated to MortgageNew Xml Schema. The well-formed document

and validated document are shown in Figure C.2. and Figure C.3. respectively. Second, the researcher checks the created Supplier-PartNew Xml Schema in well-formed. The well-formed schema is shown in Figure C.4. In addition, the researcher checks the created Supplier-PartNew XML Document in well-formed and validated to Supplier-PartNew Xml Schema. The well-formed document and validated document are shown in Figure C.5. and Figure C.6. respectively.

Second, the researcher compares file size of the XML Schemas source and file size of the created XML Schemas. See Table 5.1., in general the biggest size of XML Schemas is the Flat XML Schema normalized and the smallest size one is Hierarchical XML Schema. The reason is in Flat XML Schema the relationship among data elements are represented explicitly via key and keyref elements. In Hierarchical XML Schema, however, the relationships among data elements are represented implicitly via relationship parent and child elements (see the end of section 2.2). To define keys and keys reference need tags which increase the file size. Moreover, usually the tag is created that is understandable by people.

Third, the researcher compares file size of the XML Documents source and file size of the created XML Documents. Similar trend with XML Schemas, generally the biggest size of XML Documents is the Flat XML Documents normalized and the smallest size one is Hierarchical XML Documents. The reason in the normalized Flat XML Documents split the non normalized complex type into several normalized ones. To relate the split complex type needs duplication element which functions as a foreign key. Moreover, the split complex types need tags to define keys and keys reference. Nevertheless, in created Flat XML Documents with NIAM conceptual schema duplication data is guaranty minimum, only in the foreign key. Moreover, certain

update anomalous and inconsistent data can be omitted. Therefore, file sizes can not be used as a measure to the created software tool.

Table 5.1. The Comparison XML File.

File Name	Hierarchy XML	Non Normalized	Normalized
Mortgage-A.xml	3 KB		
Mortgage-A.xsd	2 KB		
Mortgage-B.xml	3 KB		
Mortgage-B.xsd	3 KB		
Mortgage-BNew.xml			4 KB
Mortgage-BNew.xsd			3 KB
Supplier-Part.xml		3 KB	
Supplier-Part.xsd		3 KB	
Supplier-Part-A.xml	3 KB		
Supplier-Part-A.xsd	2 KB		
Supplier-Part-B.xml	3 KB		
Supplier-Part-B.xsd	2 KB		
Supplier-Part-C.xml	4 KB		
Supplier-Part-C.xsd	2 KB		
Supplier-PartNew.xml			4 KB
Supplier-PartNew.xsd			4 KB

VI. CONCLUSION AND RECOMMENDATIONS

6.1 Conclusions

After conducting study of XML Schema design with conceptual schema approach, the researcher concludes,

- (1) The NIAM conceptual schema generally can be applied meaningfully to improve the poorly designed XML conceptual schema, except derivation role. However, it is not too significant.
- (2) The good conceptual modeling techniques for XML Schemas design have already found. Therefore, users who create XML Schema have a conceptual framework. In addition, users who want to create XML Document now have a well-structured XML Schema to follow. Furthermore, the designers who will create a XML Editor or Database application feature have the new conceptual modeling techniques for a XML Schema design. The conceptual modeling techniques are
 - (A) The maxOccurs plays the important role in hierarchical relationship definition in XML Schema. Therefore, in Hierarchical XML Schemas obtain one-to-one, many-to-one, one-to-many, and many-to-many relationship. Duplication data in child instance elements is not only for many-to-many relationship but also for many-to-one relationship. Where many-to-one relationship and duplication child instance elements for many-to-one relationship do not discuss in Hierarchical Database Model. In Hierarchical XML, the relations between the data elements are represented through the implicit hierarchical relations between parent and child elements. In addition, to mapping

Hierarchical XML Schemas to Relation Schemas using hierarchical sequence. However, the relations between the data elements are represented through the relations explicitly via key and keyref elements. The knowledge above is used to create Hierarchical XML Schemas and to transform from Hierarchical XML Schemas into Flat XML Schemas algorithms.

- (B) The CSDP procedure can be used to reverse engineering from a XML Schema into a NIAM conceptual schema. However, CSDP will be started from the fourth step by checking the uniqueness constraints. Follow by checking key length, in order to decide splitable fact type into an elementary fact type. The reverse engineering process employs meta tables. The essential coding in the meta table are Uniqueness code and MinCardinality code in Role Table because these codes play importantly in the forward engineering.
- (C) Rmap procedure can be used for transforming the NIAM conceptual schema in the meta tables into the relation schemas. Conceptually the Rmap procedure works by grouping fact type into table schema. In Rmap procedure, the minOccurs plays the important role in transforming one-to-one relationship. After that a XML Schema can be built from the created relation schemas. In addition, a XML Document can be converted to a new XML Document against with the new XML Schema.
- (3) A software tool for reengineering XML Schemas using the conceptual schema approach have already generated with several limitations. Therefore, the software tool is ready to reengineering XML Schemas which

usage is upswing nowadays.

6.2 Recommendations

There are several recommendations to solve the limitations that have already implemented as well as unimplemented, i.e.

- (1) XMLwriter can be used as XML Editors to check well-formed the XML Schemas and to check well-formed and validate the XML Documents.
- (2) Microsoft's Visual Studio .NET can be used to visualize XML Schemas and XML Documents.
- (3) To define more than one instance in a XML Document should define the forest (see Figure Listing 2.9. (a)). This definition can declare Hierarchical XML such as Figure 4.5. (c) (1) or Figure 4.6. (a).
- (4) Because VB.NET does not support attribute xsi:schemaLocation in order to connect XML documents with XML schema, load the **XML Schema** first into DataSet using **ReadXmlSchema** method. After loading **XML Document** into DataSet, use **ReadXml** method and **XmlReadMode "IgnoreSchema"**.
- (5) To fully implement stored fact type and constraint NIAM conceptual schema in XML Schema, use **XmlTextReader** and **XmlTextWriter** methods. In addition, meta tables can be used directly because the researcher creates it to capture all information.
- (6) To concern with the size of file output by means of implement derivation rule and implement constraint, first, define global element in XML Schema therefore it can be reused by ref attribute (see section 2.2 (4)). Second, by setting **ColumnMapping** property, see Table 2.7. In this study, the researcher uses the default **ColumnMapping** setting,

Column.ColumnMapping=MappingType.Element. The result is shown in Figure 4.33 (b). To reduce XML Document, set the property with **Column.ColumnMapping=MappingType.Attribute.** The result will become:

```
<Supplier Snumber="S1" Sname="Smith" City_Name="London" />
<Supplier Snumber="S2" Sname="Jones" City_Name="Paris" />
...
<Supplier Snumber="S5" Sname="Adams" City_Name="Athens" />
```

- (7) If XML Documents is less significant, then the software tool needs more participated UoD expert to decide whether the population is significant by common sense.

6.3 Suggestion for Further Research

The suggestion for further researcher is to complete the software tool with implementing the derivation rules NIAM conceptual schema in XML Schema, such as checking arithmetic derivation and entity types that should be combined, and the third step CSDP. In addition, implement the other constraints, such as lists various constraints, the sixth step CSDP. To solve the limitations of ADO.NET, **XmlTextReader** and **XmlTextWriter** methods can be used. To generate elementary fact type the other method can be used, except functional dependency. Moreover, the next research should concern with the size of file output. Moreover, the researcher suggests the free programming language, such as Java.



APPENDIX A

XML SCHEMA AND XML DOCUMENT INPUT

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3      <xsd:annotation>
4          <xsd:documentation>
5              Mortgage record XML schema.
6          </xsd:documentation>
7      </xsd:annotation>
8      <xsd:element name="Root" type="documents" />
9      <xsd:complexType name="documents">
10         <xsd:sequence>
11             <xsd:element name="document" type="documentType"
12                 minOccurs="0" maxOccurs="unbounded" />
13         </xsd:sequence>
14     </xsd:complexType>
15     <xsd:complexType name="documentType">
16         <xsd:sequence>
17             <xsd:element ref="comment" minOccurs="1" />
18             <xsd:element name="mortgagee" type="recordType" />
19             <xsd:element name="mortgage" type="mortgageType"
20                 minOccurs="0" maxOccurs="8" />
21             <xsd:element name="bank" type="recordType" />
22         </xsd:sequence>
23         <xsd:attribute name="documentDate" type="xsd:date" />
24     </xsd:complexType>
25     <xsd:complexType name="recordType">
26         <xsd:sequence>
27             <xsd:element name="name" type="xsd:string" />
28             <xsd:element name="location" type="xsd:string" />
29             <xsd:element name="city" type="xsd:string" />
30             <xsd:element name="state" type="xsd:string" minOccurs="0"/>
31         </xsd:sequence>
32         <xsd:attribute name="phone" type="xsd:string" use="optional"
33             form="qualified" />
34     </xsd:complexType>
35     <xsd:complexType name="mortgageType">
36         <xsd:sequence>
37             <xsd:element name="loanNumber" type="loanNumberType" />
38             <xsd:element name="property" type="xsd:string" />
39             <xsd:element name="date" type="xsd:date" minOccurs="0" />
40             <xsd:element name="loanAmount" type="xsd:decimal" />
41             <xsd:element name="term" type="termType" />
42         </xsd:sequence>
43     </xsd:complexType>
44     <xsd:simpleType name="loanNumberType">
45         <xsd:restriction base="xsd:string">
46             <xsd:pattern value="\d{2} \d{4} \d{2}" />
47         </xsd:restriction>
48     </xsd:simpleType>
49     <xsd:simpleType name="termType">
50         <xsd:restriction base="xsd:integer">
51             <xsd:maxInclusive value="30" />
52         </xsd:restriction>
53     </xsd:simpleType>
54     <xsd:element name="comment" type="xsd:string" />
55 </xsd:schema>

```

Figure A.1. The Listing of the Mortgage XML Schema.

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <Root xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:noNamespaceSchemaLocation="Mortgage-A.xsd" >
3   <document documentDate="2005-03-02">
4     <comment>Good risk</comment>
5     <mortgagee phone="888.555.1234">
6       <name>James Blandings</name>
7       <location>1234 299th St</location>
8       <city>New York</city>
9       <state>NY</state>
10    </mortgagee>
11    <mortgage>
12      <loanNumber>66 7777 88</loanNumber>
13      <property>The Hackett Place</property>
14      <date>2005-03-01</date>
15      <loanAmount>80000</loanAmount>
16      <term>15</term>
17    </mortgage>
18    <mortgage>
19      <loanNumber>11 8888 22</loanNumber>
20      <property>123 Acorn Drive</property>
21      <date>2005-03-01</date>
22      <loanAmount>90000</loanAmount>
23      <term>15</term>
24    </mortgage>
25    <mortgage>
26      <loanNumber>33 4444 11</loanNumber>
27      <property>99 West Pocusset St</property>
28      <date>2005-03-02</date>
29      <loanAmount>100000</loanAmount>
30      <term>30</term>
31    </mortgage>
32    <mortgage>
33      <loanNumber>55 3333 88</loanNumber>
34      <property>19 Johnson Place</property>
35      <date>2005-03-02</date>
36      <loanAmount>110000</loanAmount>
37      <term>30</term>
38    </mortgage>
39    <mortgage>
40      <loanNumber>22 6666 99</loanNumber>
41      <property>345 Nottingham Court</property>
42      <date>2005-03-02</date>
43      <loanAmount>120000</loanAmount>
44      <term>30</term>
45    </mortgage>
46    <bank phone="888.555.8888">
47      <name>XML Bank</name>
48      <location>12 Schema Place</location>
49      <city>New York</city>
50      <state>NY</state>
51    </bank>
52  </document>
53  <document documentDate="2004-07-11">
54    <comment>Good</comment>

```

Figure A.2. The Listing of the Mortgage XML Document.

```

55     <mortgagee phone="8702205">
56         <name>Widya</name>
57         <location>Manukan</location>
58         <city>Surabaya</city>
59     </mortgagee>
60     <mortgage>
61         <loanNumber>11 2233 44</loanNumber>
62         <property>Bungalow</property>
63         <date>2004-07-12</date>
64         <loanAmount>5000</loanAmount>
65         <term>12</term>
66     </mortgage>
67     <mortgage>
68         <loanNumber>11 1222 33</loanNumber>
69         <property>House</property>
70         <date>2004-12-25</date>
71         <loanAmount>3000</loanAmount>
72         <term>24</term>
73     </mortgage>
74     <mortgage>
75         <loanNumber>12 3122 34</loanNumber>
76         <property>Bungalow</property>
77         <date>2004-07-12</date>
78         <loanAmount>5000</loanAmount>
79         <term>12</term>
80     </mortgage>
81     <bank phone="8111101">
82         <name>Niaga</name>
83         <location>Tunjungan</location>
84         <city>Surabaya</city>
85     </bank>
86 </document>
87 <document documentDate="2004-07-14">
88     <comment>Good</comment>
89     <mortgagee phone="8702205">
90         <name>Widya</name>
91         <location>Manukan</location>
92         <city>Surabaya</city>
93     </mortgagee>
94     <mortgage>
95         <loanNumber>12 3122 34</loanNumber>
96         <property>Bungalow</property>
97         <date>2004-07-12</date>
98         <loanAmount>5000</loanAmount>
99         <term>12</term>
100     </mortgage>
101     <bank phone="888.555.8888">
102         <name>XML Bank</name>
103         <location>12 Schema Place</location>
104         <city>New York</city>
105         <state>NY</state>
106     </bank>
107 </document>
108 </Root>

```

Figure A.2. The Listing of the Mortgage XML Document (Continued).


```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-cem:xml-msdata">
3   <xsd:element name="NIAM" msdata:IsDataSet="true"
    msdata:EnforceConstraints="true">
4     <xsd:complexType>
5       <xsd:choice maxOccurs="unbounded">
6         <xsd:element name="Supplier">
7           <xsd:complexType>
8             <xsd:sequence>
9               <xsd:element name="Snumber" type="xsd:string" />
10              <xsd:element name="Sname" type="xsd:string" />
11              <xsd:element name="Status_Num" type="xsd:string" />
12              <xsd:element name="City_Name" type="xsd:string" />
13            </xsd:sequence>
14          </xsd:complexType>
15        </xsd:element>
16        <xsd:element name="Part">
17          <xsd:complexType>
18            <xsd:sequence>
19              <xsd:element name="Pnumber" type="xsd:string" />
20              <xsd:element name="Pname" type="xsd:string" />
21              <xsd:element name="ColorName" type="xsd:string" />
22              <xsd:element name="Loc_Num" type="xsd:string" />
23            </xsd:sequence>
24          </xsd:complexType>
25        </xsd:element>
26        <xsd:element name="Supplier_Part">
27          <xsd:complexType>
28            <xsd:sequence>
29              <xsd:element name="Snumber" type="xsd:string" />
30              <xsd:element name="Pnumber" type="xsd:string" />
31              <xsd:element name="No_of_Item" type="xsd:integer" />
32            </xsd:sequence>
33          </xsd:complexType>
34        </xsd:element>
35      </xsd:choice>
36    </xsd:complexType>
37    <xsd:key name="SupplierKey">
38      <xsd:selector xpath="."/><Supplier" />
39      <xsd:field xpath="Snumber" />
40    </xsd:key>
41    <xsd:key name="PartKey">
42      <xsd:selector xpath="."/><Part" />
43      <xsd:field xpath="Pnumber" />
44    </xsd:key>
45    <xsd:key name="Supplier_PartKey">
46      <xsd:selector xpath="."/><Supplier_Part" />
47      <xsd:field xpath="Snumber" />
48      <xsd:field xpath="Pnumber" />
49    </xsd:key>
50    <xsd:keyref name="Supplier_Supplier_Part" refer="SupplierKey">

```

Figure A.3. The Listing of the Supplier-Part XML Schema.

```

51     <xsd:selector xpath=".//Supplier_Part" />
52     <xsd:field xpath="Snumber" />
53   </xsd:keyref>
54   <xsd:keyref name="Supplier_Part_Part" refer="PartKey">
55     <xsd:selector xpath=".//Supplier_Part" />
56     <xsd:field xpath="Pnumber" />
57   </xsd:keyref>
58 </xsd:element>
59 </xsd:schema>

```

Figure A.3. The Listing of the Supplier-Part XML Schema (Continued).



```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <NIAM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Supplier-Part.xsd">
3   <Supplier>
4     <Snumber>S1</Snumber>
5     <Sname>Smith</Sname>
6     <Status_Num>20</Status_Num>
7     <City_Name>London</City_Name>
8   </Supplier>
9   <Supplier>
10    <Snumber>S2</Snumber>
11    <Sname>Jones</Sname>
12    <Status_Num>10</Status_Num>
13    <City_Name>Paris</City_Name>
14  </Supplier>
15  <Supplier>
16    <Snumber>S3</Snumber>
17    <Sname>Blake</Sname>
18    <Status_Num>10</Status_Num>
19    <City_Name>Paris</City_Name>
20  </Supplier>
21  <Supplier>
22    <Snumber>S4</Snumber>
23    <Sname>Clark</Sname>
24    <Status_Num>20</Status_Num>
25    <City_Name>London</City_Name>
26  </Supplier>
27  <Supplier>
28    <Snumber>S5</Snumber>
29    <Sname>Adams</Sname>
30    <Status_Num>30</Status_Num>
31    <City_Name>Athens</City_Name>
32  </Supplier>
33  <Part>
34    <Pnumber>P1</Pnumber>
35    <Pname>Nut</Pname>
36    <ColorName>Red</ColorName>
37    <Loc_Num>London</Loc_Num>
38  </Part>
39  <Part>
40    <Pnumber>P2</Pnumber>
41    <Pname>Bolt</Pname>
42    <ColorName>Green</ColorName>
43    <Loc_Num>Paris</Loc_Num>
44  </Part>
45  <Part>
46    <Pnumber>P3</Pnumber>
47    <Pname>Screw</Pname>
48    <ColorName>Blue</ColorName>
49    <Loc_Num>Rome</Loc_Num>
50  </Part>
51  <Part>
52    <Pnumber>P4</Pnumber>
53    <Pname>Screw</Pname>

```

Figure A.4. The Listing of the Supplier-Part XML Document.

```

54     <ColorName>Red</ColorName>
55     <Loc_Num>London</Loc_Num>
56 </Part>
57 <Part>
58     <Pnumber>P5</Pnumber>
59     <Pname>Cam</Pname>
60     <ColorName>Blue</ColorName>
61     <Loc_Num>Paris</Loc_Num>
62 </Part>
63 <Part>
64     <Pnumber>P6</Pnumber>
65     <Pname>Cog</Pname>
66     <ColorName>Red</ColorName>
67     <Loc_Num>London</Loc_Num>
68 </Part>
69 <Supplier_Part>
70     <Snumber>S1</Snumber>
71     <Pnumber>P1</Pnumber>
72     <No_of_Item>300</No_of_Item>
73 </Supplier_Part>
74 <Supplier_Part>
75     <Snumber>S1</Snumber>
76     <Pnumber>P2</Pnumber>
77     <No_of_Item>200</No_of_Item>
78 </Supplier_Part>
79 <Supplier_Part>
80     <Snumber>S1</Snumber>
81     <Pnumber>P3</Pnumber>
82     <No_of_Item>400</No_of_Item>
83 </Supplier_Part>
84 <Supplier_Part>
85     <Snumber>S1</Snumber>
86     <Pnumber>P4</Pnumber>
87     <No_of_Item>200</No_of_Item>
88 </Supplier_Part>
89 <Supplier_Part>
90     <Snumber>S1</Snumber>
91     <Pnumber>P5</Pnumber>
92     <No_of_Item>100</No_of_Item>
93 </Supplier_Part>
94 <Supplier_Part>
95     <Snumber>S1</Snumber>
96     <Pnumber>P6</Pnumber>
97     <No_of_Item>100</No_of_Item>
98 </Supplier_Part>
99 <Supplier_Part>
100     <Snumber>S2</Snumber>
101     <Pnumber>P1</Pnumber>
102     <No_of_Item>300</No_of_Item>
103 </Supplier_Part>
104 <Supplier_Part>
105     <Snumber>S2</Snumber>
106     <Pnumber>P2</Pnumber>
107     <No_of_Item>400</No_of_Item>
108 </Supplier_Part>

```

Figure A.4. The Listing of the Supplier-Part XML Document (Continued).

```

109 <Supplier_Part>
110   <Snumber>S3</Snumber>
111   <Pnumber>P2</Pnumber>
112   <No_of_Item>200</No_of_Item>
113 </Supplier_Part>
114 <Supplier_Part>
115   <Snumber>S4</Snumber>
116   <Pnumber>P2</Pnumber>
117   <No_of_Item>200</No_of_Item>
118 </Supplier_Part>
119 <Supplier_Part>
120   <Snumber>S4</Snumber>
121   <Pnumber>P4</Pnumber>
122   <No_of_Item>300</No_of_Item>
123 </Supplier_Part>
124 <Supplier_Part>
125   <Snumber>S4</Snumber>
126   <Pnumber>P5</Pnumber>
127   <No_of_Item>400</No_of_Item>
128 </Supplier_Part>
129 </NIAM>

```

Figure A.4. The Listing of the Supplier-Part XML Document (Continued).



```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <xsd:element name="Root" type="Supply_Part_DataBases" />
4   <xsd:complexType name="Supply_Part_DataBases">
5     <xsd:sequence>
6       <xsd:element name="Supplier" type="Supply_PartType"
7         maxOccurs="unbounded" />
8     </xsd:sequence>
9   </xsd:complexType>
10  <xsd:complexType name="Supply_PartType">
11    <xsd:sequence>
12      <xsd:element name="Sname" type="xsd:string" />
13      <xsd:element name="Status_Num" type="xsd:string" />
14      <xsd:element name="City_Name" type="xsd:string" />
15      <xsd:element name="Part" type="PartType" minOccurs="0"
16        maxOccurs="unbounded" />
17    </xsd:sequence>
18    <xsd:attribute name="Snumber" type="xsd:string" />
19  </xsd:complexType>
20  <xsd:complexType name="PartType">
21    <xsd:sequence>
22      <xsd:element name="Pname" type="xsd:string" />
23      <xsd:element name="ColorName" type="xsd:string" />
24      <xsd:element name="Loc_Num" type="xsd:string" />
25      <xsd:element name="No_of_Item" type="xsd:integer" />
26    </xsd:sequence>
27    <xsd:attribute name="Pnumber" type="xsd:string" />
28  </xsd:complexType>
29 </xsd:schema>

```

Figure A.5. The Listing of the Supplier-Part-A XML Schema.


```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Supplier-Part-A.xsd">
3   <Supplier Snumber="S1">
4     <Sname>Smith</Sname>
5     <Status_Num>20</Status_Num>
6     <City_Name>London</City_Name>
7     <Part Pnumber="P1">
8       <Pname>Nut</Pname>
9       <ColorName>Red</ColorName>
10      <Loc_Num>London</Loc_Num>
11      <No_of_Item>300</No_of_Item>
12    </Part>
13    <Part Pnumber="P2">
14      <Pname>Bolt</Pname>
15      <ColorName>Green</ColorName>
16      <Loc_Num>Paris</Loc_Num>
17      <No_of_Item>200</No_of_Item>
18    </Part>
19    <Part Pnumber="P3">
20      <Pname>Screw</Pname>
21      <ColorName>Blue</ColorName>
22      <Loc_Num>Rome</Loc_Num>
23      <No_of_Item>400</No_of_Item>
24    </Part>
25    <Part Pnumber="P4">
26      <Pname>Screw</Pname>
27      <ColorName>Red</ColorName>
28      <Loc_Num>London</Loc_Num>
29      <No_of_Item>200</No_of_Item>
30    </Part>
31    <Part Pnumber="P5">
32      <Pname>Cam</Pname>
33      <ColorName>Blue</ColorName>
34      <Loc_Num>Paris</Loc_Num>
35      <No_of_Item>100</No_of_Item>
36    </Part>
37    <Part Pnumber="P6">
38      <Pname>Cog</Pname>
39      <ColorName>Red</ColorName>
40      <Loc_Num>London</Loc_Num>
41      <No_of_Item>100</No_of_Item>
42    </Part>
43  </Supplier>
44  <Supplier Snumber="S2">
45    <Sname>Jones</Sname>
46    <Status_Num>10</Status_Num>
47    <City_Name>Paris</City_Name>
48    <Part Pnumber="P1">
49      <Pname>Nut</Pname>
50      <ColorName>Red</ColorName>
51      <Loc_Num>London</Loc_Num>
52      <No_of_Item>300</No_of_Item>
53    </Part>

```

Figure A.6. The Listing of the Supplier-Part-A XML Document.

```

54     <Part Pnumber="P2">
55         <Pname>Bolt</Pname>
56         <ColorName>Green</ColorName>
57         <Loc_Num>Paris</Loc_Num>
58         <No_of_Item>400</No_of_Item>
59     </Part>
60 </Supplier>
61 <Supplier Snumber="S3">
62     <Sname>Blake</Sname>
63     <Status_Num>10</Status_Num>
64     <City_Name>Paris</City_Name>
65     <Part Pnumber="P2">
66         <Pname>Bolt</Pname>
67         <ColorName>Green</ColorName>
68         <Loc_Num>Paris</Loc_Num>
69         <No_of_Item>200</No_of_Item>
70     </Part>
71 </Supplier>
72 <Supplier Snumber="S4">
73     <Sname>Clark</Sname>
74     <Status_Num>20</Status_Num>
75     <City_Name>London</City_Name>
76     <Part Pnumber="P2">
77         <Pname>Bolt</Pname>
78         <ColorName>Green</ColorName>
79         <Loc_Num>Paris</Loc_Num>
80         <No_of_Item>200</No_of_Item>
81     </Part>
82     <Part Pnumber="P4">
83         <Pname>Screw</Pname>
84         <ColorName>Red</ColorName>
85         <Loc_Num>London</Loc_Num>
86         <No_of_Item>300</No_of_Item>
87     </Part>
88     <Part Pnumber="P5">
89         <Pname>Cam</Pname>
90         <ColorName>Blue</ColorName>
91         <Loc_Num>Paris</Loc_Num>
92         <No_of_Item>400</No_of_Item>
93     </Part>
94 </Supplier>
95 <Supplier Snumber="S5">
96     <Sname>Adams</Sname>
97     <Status_Num>30</Status_Num>
98     <City_Name>Athens</City_Name>
99 </Supplier>
100 </Root>

```

Figure A.6. The Listing of the Supplier-Part-A XML Document (Continued).

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <xsd:element name="Root" type="Supply_Part_DataBases" />
4   <xsd:complexType name="Supply_Part_DataBases">
5     <xsd:sequence>
6       <xsd:element name="Supplier" type="Supply_PartType"
7         maxOccurs="unbounded" />
8     </xsd:sequence>
9   </xsd:complexType>
10  <xsd:complexType name="Supply_PartType">
11    <xsd:sequence>
12      <xsd:element name="Sname" type="xsd:string" />
13      <xsd:element name="Status_Num" type="xsd:string" />
14      <xsd:element name="City_Name" type="xsd:string" />
15      <xsd:element name="Supplier_Part" type="Supplier_PartType"
16        minOccurs="0" maxOccurs="unbounded" />
17    </xsd:sequence>
18    <xsd:attribute name="Snumber" type="xsd:string" />
19  </xsd:complexType>
20  <xsd:complexType name="Supplier_PartType">
21    <xsd:sequence>
22      <xsd:element name="Part">
23        <xsd:complexType>
24          <xsd:sequence>
25            <xsd:element name="Pname" type="xsd:string" />
26            <xsd:element name="ColorName" type="xsd:string" />
27            <xsd:element name="Loc_Num" type="xsd:string" />
28          </xsd:sequence>
29          <xsd:attribute name="Pnumber" type="xsd:string" />
30        </xsd:complexType>
31      </xsd:element>
32      <xsd:element name="No_of_Item" type="xsd:integer" />
33    </xsd:sequence>
34  </xsd:complexType>
35 </xsd:schema>

```

Figure A.7. The Listing of the Supplier-Part-C XML Schema.

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Supplier-Part-C.xsd">
3   <Supplier Snumber="S1">
4     <Sname>Smith</Sname>
5     <Status_Num>20</Status_Num>
6     <City_Name>London</City_Name>
7     <Supplier_Part>
8       <Part Pnumber="P1">
9         <Pname>Nut</Pname>
10        <ColorName>Red</ColorName>
11        <Loc_Num>London</Loc_Num>
12      </Part>
13      <No_of_Item>300</No_of_Item>
14    </Supplier_Part>
15    <Supplier_Part>
16      <Part Pnumber="P2">
17        <Pname>Bolt</Pname>
18        <ColorName>Green</ColorName>
19        <Loc_Num>Paris</Loc_Num>
20      </Part>
21      <No_of_Item>200</No_of_Item>
22    </Supplier_Part>
23    <Supplier_Part>
24      <Part Pnumber="P3">
25        <Pname>Screw</Pname>
26        <ColorName>Blue</ColorName>
27        <Loc_Num>Rome</Loc_Num>
28      </Part>
29      <No_of_Item>400</No_of_Item>
30    </Supplier_Part>
31    <Supplier_Part>
32      <Part Pnumber="P4">
33        <Pname>Screw</Pname>
34        <ColorName>Red</ColorName>
35        <Loc_Num>London</Loc_Num>
36      </Part>
37      <No_of_Item>200</No_of_Item>
38    </Supplier_Part>
39    <Supplier_Part>
40      <Part Pnumber="P5">
41        <Pname>Cam</Pname>
42        <ColorName>Blue</ColorName>
43        <Loc_Num>Paris</Loc_Num>
44      </Part>
45      <No_of_Item>100</No_of_Item>
46    </Supplier_Part>
47    <Supplier_Part>
48      <Part Pnumber="P6">
49        <Pname>Cog</Pname>
50        <ColorName>Red</ColorName>
51        <Loc_Num>London</Loc_Num>
52      </Part>
53      <No_of_Item>100</No_of_Item>

```

Figure A.8. The Listing of the Supplier-Part-C XML Document.

```

54     </Supplier_Part>
55 </Supplier>
56 <Supplier Snumber="S2">
57     <Sname>Jones</Sname>
58     <Status_Num>10</Status_Num>
59     <City_Name>Paris</City_Name>
60     <Supplier_Part>
61         <Part Pnumber="P1">
62             <Pname>Nut</Pname>
63             <ColorName>Red</ColorName>
64             <Loc_Num>London</Loc_Num>
65         </Part>
66         <No_of_Item>300</No_of_Item>
67     </Supplier_Part>
68     <Supplier_Part>
69         <Part Pnumber="P2">
70             <Pname>Bolt</Pname>
71             <ColorName>Green</ColorName>
72             <Loc_Num>Paris</Loc_Num>
73         </Part>
74         <No_of_Item>400</No_of_Item>
75     </Supplier_Part>
76 </Supplier>
77 <Supplier Snumber="S3">
78     <Sname>Blake</Sname>
79     <Status_Num>10</Status_Num>
80     <City_Name>Paris</City_Name>
81     <Supplier_Part>
82         <Part Pnumber="P2">
83             <Pname>Bolt</Pname>
84             <ColorName>Green</ColorName>
85             <Loc_Num>Paris</Loc_Num>
86         </Part>
87         <No_of_Item>200</No_of_Item>
88     </Supplier_Part>
89 </Supplier>
90 <Supplier Snumber="S4">
91     <Sname>Clark</Sname>
92     <Status_Num>20</Status_Num>
93     <City_Name>London</City_Name>
94     <Supplier_Part>
95         <Part Pnumber="P2">
96             <Pname>Bolt</Pname>
97             <ColorName>Green</ColorName>
98             <Loc_Num>Paris</Loc_Num>
99         </Part>
100         <No_of_Item>200</No_of_Item>
101     </Supplier_Part>
102     <Supplier_Part>
103         <Part Pnumber="P4">
104             <Pname>Screw</Pname>
105             <ColorName>Red</ColorName>
106             <Loc_Num>London</Loc_Num>
107         </Part>
108         <No_of_Item>300</No_of_Item>

```

Figure A.8. The Listing of the Supplier-Part-C XML Document (Continued).

```

109     </Supplier_Part>
110     <Supplier_Part>
111         <Part Pnumber="P5">
112             <Pname>Cam</Pname>
113             <ColorName>Blue</ColorName>
114             <Loc_Num>Paris</Loc_Num>
115         </Part>
116         <No_of_Item>400</No_of_Item>
117     </Supplier_Part>
118 </Supplier>
119 <Supplier Snumber="S5">
120     <Sname>Adams</Sname>
121     <Status_Num>30</Status_Num>
122     <City_Name>Athens</City_Name>
123 </Supplier>
124 </Root>

```

Figure A.8. The Listing of the Supplier-Part-C XML Document (Continued).





APPENDIX B

XML SCHEMA AND XML DOCUMENT OUTPUT

```

1 <?xml version="1.0" ?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
3   <xsd:element name="NIAM" msdata:IsDataSet="true"
    msdata:EnforceConstraints="true">
4     <xsd:complexType>
5       <xsd:choice maxOccurs="unbounded">
6         <xsd:element name="document">
7           <xsd:complexType>
8             <xsd:sequence>
9               <xsd:element name="comment" type="xsd:string" />
10              <xsd:element name="documentDate" type="xsd:date" />
11              <xsd:element name="name" type="xsd:string" />
12              <xsd:element name="bankname" type="xsd:string" />
13            </xsd:sequence>
14          </xsd:complexType>
15        </xsd:element>
16        <xsd:element name="mortgagee">
17          <xsd:complexType>
18            <xsd:sequence>
19              <xsd:element name="name" type="xsd:string" />
20              <xsd:element name="location" type="xsd:string" />
21              <xsd:element name="city" type="xsd:string" />
22              <xsd:element name="state" type="xsd:string"
23                minOccurs="0" />
24              <xsd:element name="phone" type="xsd:string" />
25            </xsd:sequence>
26          </xsd:complexType>
27        </xsd:element>
28        <xsd:element name="bank">
29          <xsd:complexType>
30            <xsd:sequence>
31              <xsd:element name="name" type="xsd:string" />
32              <xsd:element name="location" type="xsd:string" />
33              <xsd:element name="city" type="xsd:string" />
34              <xsd:element name="state" type="xsd:string"
35                minOccurs="0" />
36              <xsd:element name="phone" type="xsd:string" />
37            </xsd:sequence>
38          </xsd:complexType>
39        </xsd:element>
40        <xsd:element name="mortgage">
41          <xsd:complexType>
42            <xsd:sequence>
43              <xsd:element name="property" type="xsd:string" />
44              <xsd:element name="date" type="xsd:date" />
45              <xsd:element name="loanAmount" type="xsd:decimal" />
46              <xsd:element name="term" type="xsd:integer" />
47              <xsd:element name="loanNumber" type="xsd:string" />
48              <xsd:element name="documentDate" type="xsd:date" />
49            </xsd:sequence>
50          </xsd:complexType>
51        </xsd:element>
52      </xsd:choice>

```

Figure B.1. The Listing of the MortgageNew XML Schema.

```

51 </xsd:complexType>
52 <xsd:key name="documentKey">
53   <xsd:selector xpath="."//document" />
54   <xsd:field xpath="documentDate" />
55 </xsd:key>
56 <xsd:key name="mortgageeKey">
57   <xsd:selector xpath="."//mortgagee" />
58   <xsd:field xpath="name" />
59 </xsd:key>
60 <xsd:key name="bankKey">
61   <xsd:selector xpath="."//bank" />
62   <xsd:field xpath="name" />
63 </xsd:key>
64 <xsd:key name="mortgageKey">
65   <xsd:selector xpath="."//mortgage" />
66   <xsd:field xpath="loanNumber" />
67 </xsd:key>
68 <xsd:keyref name="document_mortgagee" refer="mortgageeKey">
69   <xsd:selector xpath="."//document" />
70   <xsd:field xpath="name" />
71 </xsd:keyref>
72 <xsd:keyref name="document_bank" refer="bankKey">
73   <xsd:selector xpath="."//document" />
74   <xsd:field xpath="bankname" />
75 </xsd:keyref>
76 <xsd:keyref name="mortgage_document" refer="documentKey">
77   <xsd:selector xpath="."//mortgage" />
78   <xsd:field xpath="documentDate" />
79 </xsd:keyref>
80 </xsd:element>
81 </xsd:schema>

```

Figure B.1. The Listing of the MortgageNew XML Schema (Continued).

```

1  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
2  <NIAM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Mortgage-BNew.xsd">
3    <document>
4      <comment>Good risk</comment>
5      <documentDate>2005-03-02</documentDate>
6      <name>James Blandings</name>
7      <bankname>XML Bank</bankname>
8    </document>
9    <document>
10     <comment>Good</comment>
11     <documentDate>2004-07-11</documentDate>
12     <name>Hans Schmidt</name>
13     <bankname>Niaga</bankname>
14   </document>
15   <document>
16     <comment>Good</comment>
17     <documentDate>2004-07-14</documentDate>
18     <name>Hans Schmidt</name>
19     <bankname>XML Bank</bankname>
20   </document>
21   <mortgagee>
22     <name>James Blandings</name>
23     <location>1234 299th St</location>
24     <city>New York</city>
25     <state>NY</state>
26     <phone>888.555.1234</phone>
27   </mortgagee>
28   <mortgagee>
29     <name>Hans Schmidt</name>
30     <location>123 Hallgarten</location>
31     <city>Berlin</city>
32     <phone>870.220.5678</phone>
33   </mortgagee>
34   <bank>
35     <name>XML Bank</name>
36     <location>12 Schema Place</location>
37     <city>New York</city>
38     <state>NY</state>
39     <phone>888.555.8888</phone>
40   </bank>
41   <bank>
42     <name>Niaga</name>
43     <location>56 Sweet Street</location>
44     <city>Berlin</city>
45     <phone>811.110.1234</phone>
46   </bank>
47   <mortgage>
48     <property>The Hackett Place</property>
49     <date>2005-03-01</date>
50     <loanAmount>80000</loanAmount>
51     <term>15</term>
52     <loanNumber>66 7777 88</loanNumber>
53     <documentDate>2005-03-02</documentDate>
54   </mortgage>

```

Figure B.2. The Listing of the MortgageNew XML Document.

```

55 <mortgage>
56   <property>123 Acorn Drive</property>
57   <date>2005-03-01</date>
58   <loanAmount>90000</loanAmount>
59   <term>15</term>
60   <loanNumber>11 8888 22</loanNumber>
61   <documentDate>2005-03-02</documentDate>
62 </mortgage>
63 <mortgage>
64   <property>99 West Pocusset St</property>
65   <date>2005-03-02</date>
66   <loanAmount>100000</loanAmount>
67   <term>30</term>
68   <loanNumber>33 4444 11</loanNumber>
69   <documentDate>2005-03-02</documentDate>
70 </mortgage>
71 <mortgage>
72   <property>19 Johnson Place</property>
73   <date>2005-03-02</date>
74   <loanAmount>110000</loanAmount>
75   <term>30</term>
76   <loanNumber>55 3333 88</loanNumber>
77   <documentDate>2005-03-02</documentDate>
78 </mortgage>
79 <mortgage>
80   <property>345 Nottingham Court</property>
81   <date>2005-03-02</date>
82   <loanAmount>120000</loanAmount>
83   <term>30</term>
84   <loanNumber>22 6666 99</loanNumber>
85   <documentDate>2005-03-02</documentDate>
86 </mortgage>
87 <mortgage>
88   <property>Bungalow</property>
89   <date>2004-07-12</date>
90   <loanAmount>55000</loanAmount>
91   <term>12</term>
92   <loanNumber>11 2233 44</loanNumber>
93   <documentDate>2004-07-11</documentDate>
94 </mortgage>
95 <mortgage>
96   <property>House</property>
97   <date>2004-12-25</date>
98   <loanAmount>95000</loanAmount>
99   <term>24</term>
100   <loanNumber>11 1222 33</loanNumber>
101   <documentDate>2004-07-11</documentDate>
102 </mortgage>
103 <mortgage>
104   <property>Bungalow</property>
105   <date>2004-07-12</date>
106   <loanAmount>50000</loanAmount>
107   <term>12</term>
108   <loanNumber>12 3122 34</loanNumber>
109   <documentDate>2004-07-14</documentDate>
110 </mortgage>
111 </NIAM>

```

Figure B.2. The Listing of the MortgageNew XML Document (Continued).

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
3    <xsd:element name="NIAM" msdata:IsDataSet="true"
   msdata:EnforceConstraints="true">
4      <xsd:complexType>
5        <xsd:choice maxOccurs="unbounded">
6          <xsd:element name="Supplier">
7            <xsd:complexType>
8              <xsd:sequence>
9                <xsd:element name="Snumber" type="xsd:string" />
10               <xsd:element name="Sname" type="xsd:string" />
11               <xsd:element name="City_Name" type="xsd:string" />
12             </xsd:sequence>
13           </xsd:complexType>
14         </xsd:element>
15       <xsd:element name="Part">
16         <xsd:complexType>
17           <xsd:sequence>
18             <xsd:element name="Pnumber" type="xsd:string" />
19             <xsd:element name="Pname" type="xsd:string" />
20             <xsd:element name="ColorName" type="xsd:string" />
21           </xsd:sequence>
22         </xsd:complexType>
23       </xsd:element>
24     <xsd:element name="City">
25       <xsd:complexType>
26         <xsd:sequence>
27           <xsd:element name="City_Name" type="xsd:string" />
28           <xsd:element name="Status_Num" type="xsd:string" />
29         </xsd:sequence>
30       </xsd:complexType>
31     </xsd:element>
32   <xsd:element name="Supplier_Part">
33     <xsd:complexType>
34       <xsd:sequence>
35         <xsd:element name="Snumber" type="xsd:string" />
36         <xsd:element name="Pnumber" type="xsd:string" />
37         <xsd:element name="No_of_Item" type="xsd:integer" />
38       </xsd:sequence>
39     </xsd:complexType>
40   </xsd:element>
41 <xsd:element name="P_Loc">
42   <xsd:complexType>
43     <xsd:sequence>
44       <xsd:element name="Pnumber" type="xsd:string" />
45       <xsd:element name="Loc_Num" type="xsd:string" />
46     </xsd:sequence>
47   </xsd:complexType>
48 </xsd:element>
49 </xsd:choice>
50 </xsd:complexType>
51 <xsd:key name="SupplierKey">
52   <xsd:selector xpath="./Supplier" />
53   <xsd:field xpath="Snumber" />

```

Figure B.3. The Listing of the Supplier-PartNew XML Schema.


```

54     </xsd:key>
55     <xsd:key name="PartKey">
56         <xsd:selector xpath="//Part" />
57         <xsd:field xpath="Pnumber" />
58     </xsd:key>
59     <xsd:key name="CityKey">
60         <xsd:selector xpath="//City" />
61         <xsd:field xpath="City_Name" />
62     </xsd:key>
63     <xsd:key name="Supplier_PartKey">
64         <xsd:selector xpath="//Supplier_Part" />
65         <xsd:field xpath="Snumber" />
66         <xsd:field xpath="Pnumber" />
67     </xsd:key>
68     <xsd:unique name="P_LocKey">
69         <xsd:selector xpath="//P_Loc" />
70         <xsd:field xpath="Pnumber" />
71         <xsd:field xpath="Loc_Num" />
72     </xsd:unique>
73     <xsd:keyref name="Supplier_Supplier_Part" refer="SupplierKey">
74         <xsd:selector xpath="//Supplier_Part" />
75         <xsd:field xpath="Snumber" />
76     </xsd:keyref>
77     <xsd:keyref name="Supplier_Part_Part" refer="PartKey">
78         <xsd:selector xpath="//Supplier_Part" />
79         <xsd:field xpath="Pnumber" />
80     </xsd:keyref>
81     <xsd:keyref name="Supplier_City" refer="CityKey">
82         <xsd:selector xpath="//Supplier" />
83         <xsd:field xpath="City_Name" />
84     </xsd:keyref>
85     <xsd:keyref name="P_Loc_Part" refer="PartKey">
86         <xsd:selector xpath="//P_Loc" />
87         <xsd:field xpath="Pnumber" />
88     </xsd:keyref>
89 </xsd:element>
90 </xsd:schema>

```

Figure B.3. The Listing of the Supplier-PartNew XML Schema (Continued).

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <NIAM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="Supplier-PartNew.xsd">
3   <Supplier>
4     <Snumber>S1</Snumber>
5     <Sname>Smith</Sname>
6     <City_Name>London</City_Name>
7   </Supplier>
8   <Supplier>
9     <Snumber>S2</Snumber>
10    <Sname>Jones</Sname>
11    <City_Name>Paris</City_Name>
12  </Supplier>
13  <Supplier>
14    <Snumber>S3</Snumber>
15    <Sname>Blake</Sname>
16    <City_Name>Paris</City_Name>
17  </Supplier>
18  <Supplier>
19    <Snumber>S4</Snumber>
20    <Sname>Clark</Sname>
21    <City_Name>London</City_Name>
22  </Supplier>
23  <Supplier>
24    <Snumber>S5</Snumber>
25    <Sname>Adams</Sname>
26    <City_Name>Athens</City_Name>
27  </Supplier>
28  <Part>
29    <Pnumber>P1</Pnumber>
30    <Pname>Nut</Pname>
31    <ColorName>Red</ColorName>
32  </Part>
33  <Part>
34    <Pnumber>P2</Pnumber>
35    <Pname>Bolt</Pname>
36    <ColorName>Green</ColorName>
37  </Part>
38  <Part>
39    <Pnumber>P3</Pnumber>
40    <Pname>Screw</Pname>
41    <ColorName>Blue</ColorName>
42  </Part>
43  <Part>
44    <Pnumber>P4</Pnumber>
45    <Pname>Screw</Pname>
46    <ColorName>Red</ColorName>
47  </Part>
48  <Part>
49    <Pnumber>P5</Pnumber>
50    <Pname>Cam</Pname>
51    <ColorName>Blue</ColorName>
52  </Part>
53  <Part>

```

Figure B.4. The Listing of the Supplier-PartNew XML Document.

```

54     <Pnumber>P6</Pnumber>
55     <Pname>Cog</Pname>
56     <ColorName>Red</ColorName>
57 </Part>
58 <City>
59     <City_Name>London</City_Name>
60     <Status_Num>20</Status_Num>
61 </City>
62 <City>
63     <City_Name>Paris</City_Name>
64     <Status_Num>10</Status_Num>
65 </City>
66 <City>
67     <City_Name>Athens</City_Name>
68     <Status_Num>30</Status_Num>
69 </City>
70 <Supplier_Part>
71     <Snumber>S1</Snumber>
72     <Pnumber>P1</Pnumber>
73     <No_of_Item>300</No_of_Item>
74 </Supplier_Part>
75 <Supplier_Part>
76     <Snumber>S1</Snumber>
77     <Pnumber>P2</Pnumber>
78     <No_of_Item>200</No_of_Item>
79 </Supplier_Part>
80 <Supplier_Part>
81     <Snumber>S1</Snumber>
82     <Pnumber>P3</Pnumber>
83     <No_of_Item>400</No_of_Item>
84 </Supplier_Part>
85 <Supplier_Part>
86     <Snumber>S1</Snumber>
87     <Pnumber>P4</Pnumber>
88     <No_of_Item>200</No_of_Item>
89 </Supplier_Part>
90 <Supplier_Part>
91     <Snumber>S1</Snumber>
92     <Pnumber>P5</Pnumber>
93     <No_of_Item>100</No_of_Item>
94 </Supplier_Part>
95 <Supplier_Part>
96     <Snumber>S1</Snumber>
97     <Pnumber>P6</Pnumber>
98     <No_of_Item>100</No_of_Item>
99 </Supplier_Part>
100 <Supplier_Part>
101     <Snumber>S2</Snumber>
102     <Pnumber>P1</Pnumber>
103     <No_of_Item>300</No_of_Item>
104 </Supplier_Part>
105 <Supplier_Part>
106     <Snumber>S2</Snumber>
107     <Pnumber>P2</Pnumber>
108     <No_of_Item>400</No_of_Item>

```

Figure B.4. The Listing of the Supplier-PartNew XML Document (Continued).

```

109 </Supplier_Part>
110 <Supplier_Part>
111   <Snumber>S3</Snumber>
112   <Pnumber>P2</Pnumber>
113   <No_of_Item>200</No_of_Item>
114 </Supplier_Part>
115 <Supplier_Part>
116   <Snumber>S4</Snumber>
117   <Pnumber>P2</Pnumber>
118   <No_of_Item>200</No_of_Item>
119 </Supplier_Part>
120 <Supplier_Part>
121   <Snumber>S4</Snumber>
122   <Pnumber>P4</Pnumber>
123   <No_of_Item>300</No_of_Item>
124 </Supplier_Part>
125 <Supplier_Part>
126   <Snumber>S4</Snumber>
127   <Pnumber>P5</Pnumber>
128   <No_of_Item>400</No_of_Item>
129 </Supplier_Part>
130 <P_Loc>
131   <Pnumber>P1</Pnumber>
132   <Loc_Num>London</Loc_Num>
133 </P_Loc>
134 <P_Loc>
135   <Pnumber>P2</Pnumber>
136   <Loc_Num>Paris</Loc_Num>
137 </P_Loc>
138 <P_Loc>
139   <Pnumber>P3</Pnumber>
140   <Loc_Num>Rome</Loc_Num>
141 </P_Loc>
142 <P_Loc>
143   <Pnumber>P4</Pnumber>
144   <Loc_Num>London</Loc_Num>
145 </P_Loc>
146 <P_Loc>
147   <Pnumber>P5</Pnumber>
148   <Loc_Num>Paris</Loc_Num>
149 </P_Loc>
150 <P_Loc>
151   <Pnumber>P6</Pnumber>
152   <Loc_Num>London</Loc_Num>
153 </P_Loc>
154 </NIAM>

```

Figure B.4. The Listing of the Supplier-PartNew XML Document (Continued).

APPENDIX C

**CHECKED WELL-XML SCHEMA OUTPUT AND CHECKED WELL-FORMED
AND VALIDATED XML DOCUMENT OUTPUT**



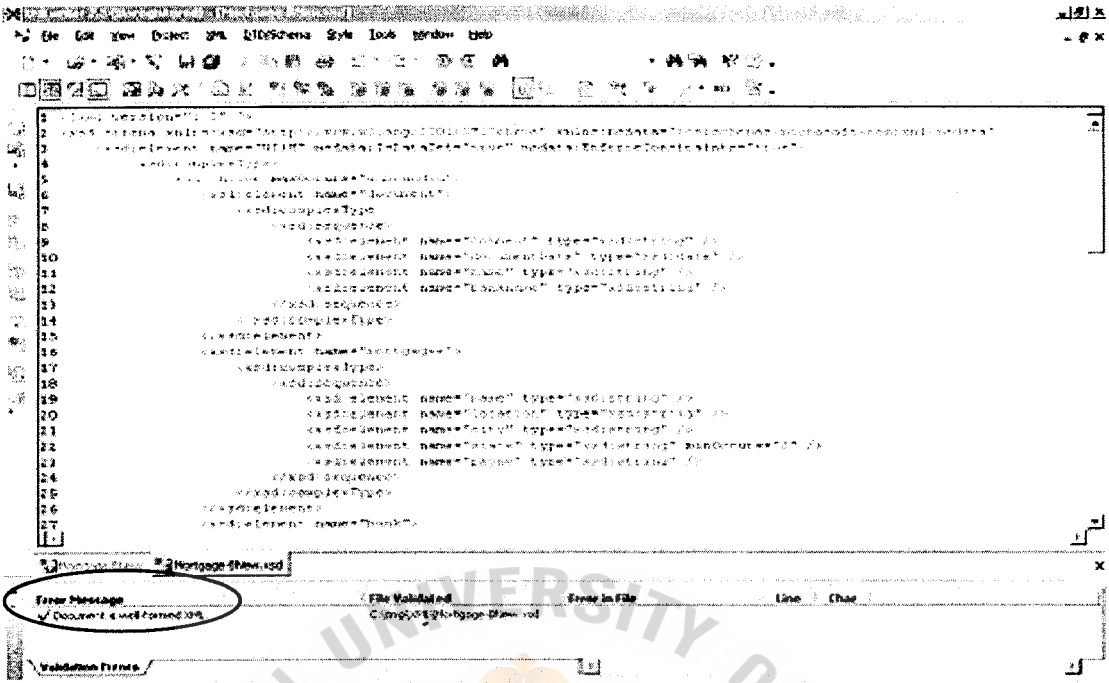


Figure C.1. Checked Well-Formed MortgageNew Xml Schema Output.

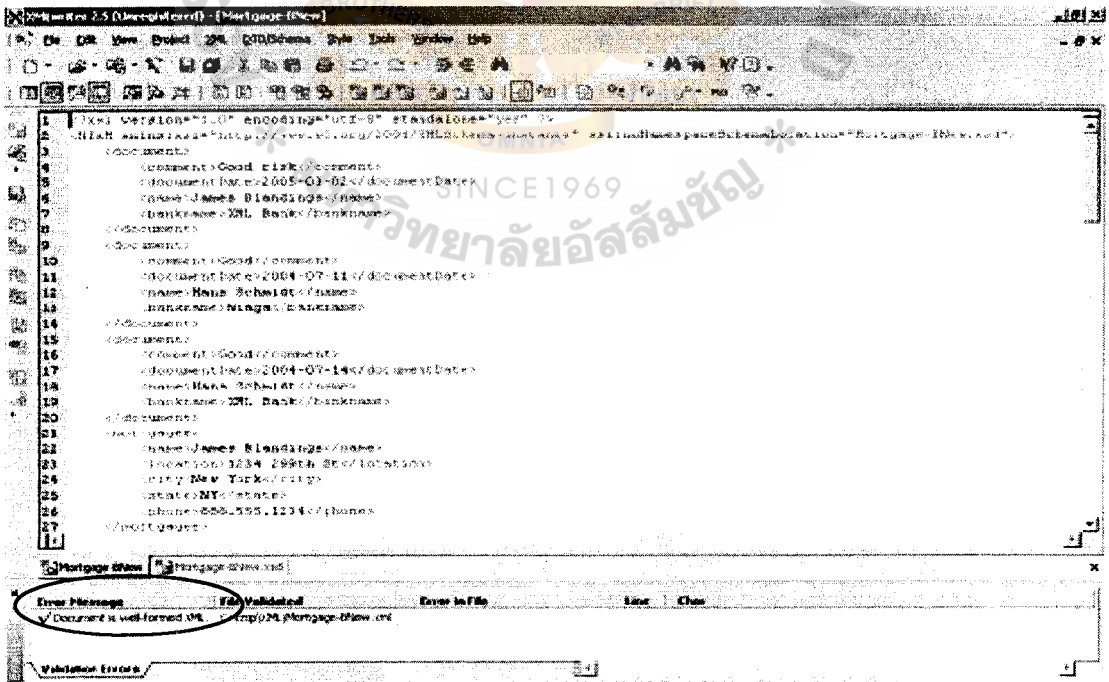


Figure C.2. Checked Well-Formed MortgageNew XML Document Output.

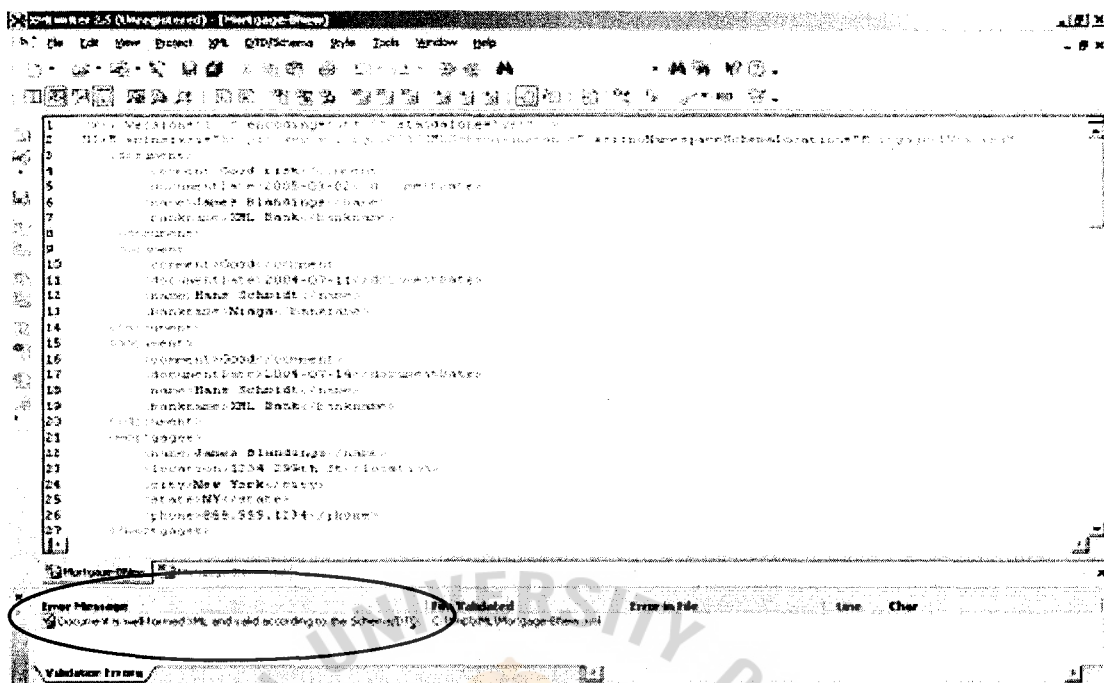


Figure C.3. Checked Validated MortgageNew Xml Document Output.

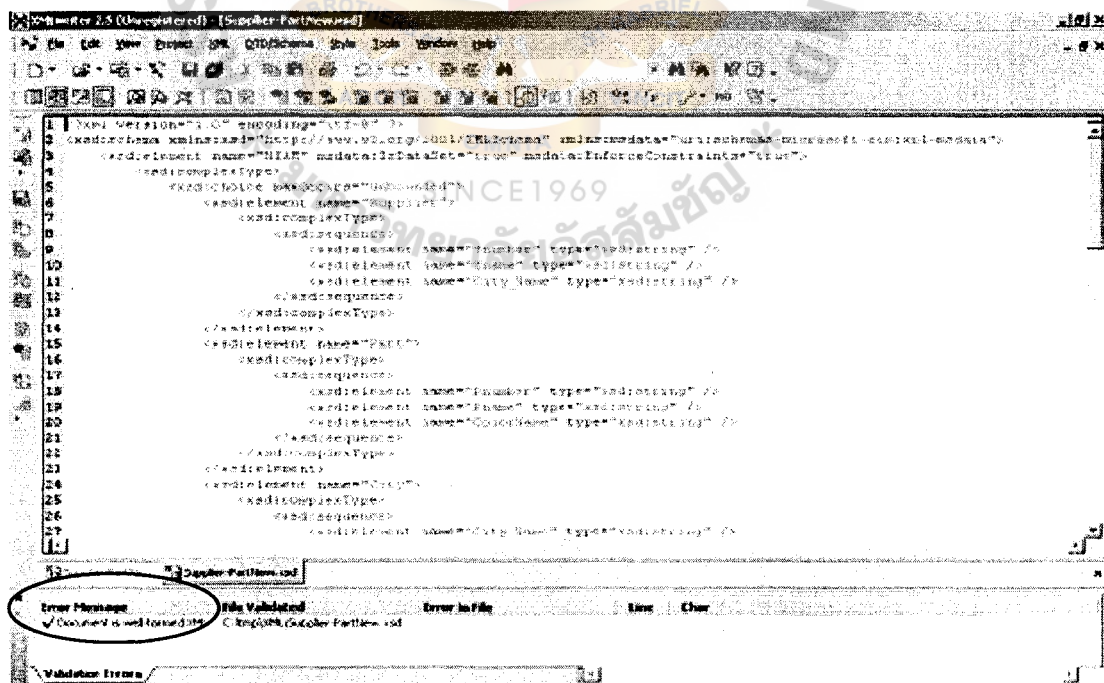


Figure C.4. Checked Well-Formed Supplier–PartNew Xml Schema Output.

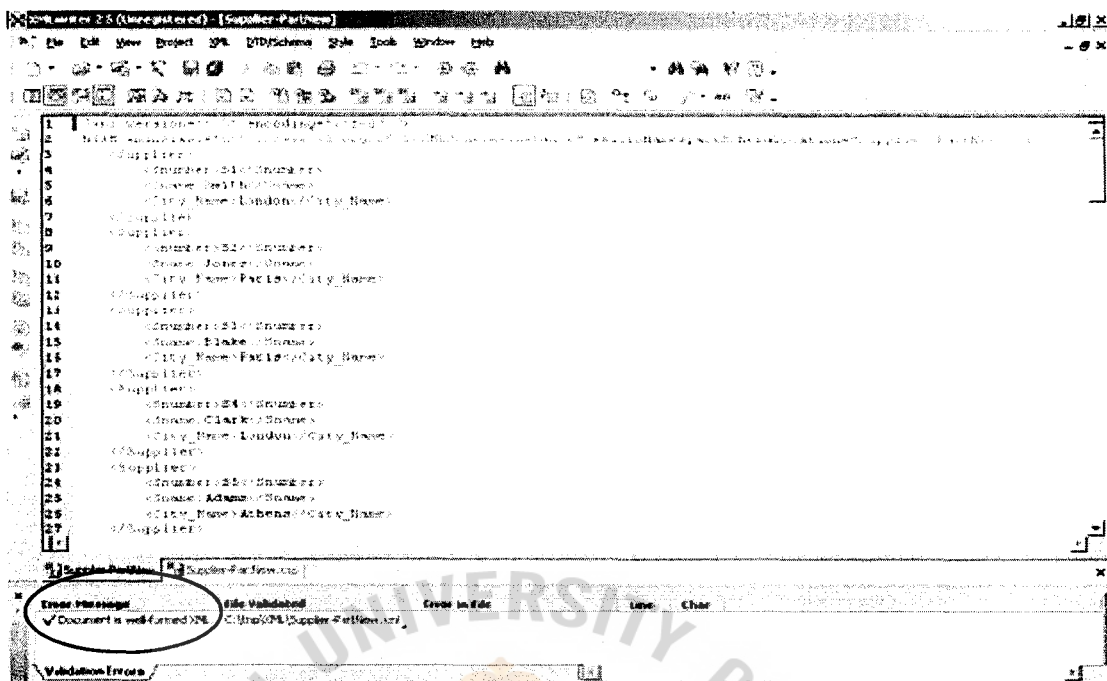


Figure C.5. Checked Well-Formed Supplier-PartNew Xml Document Output.

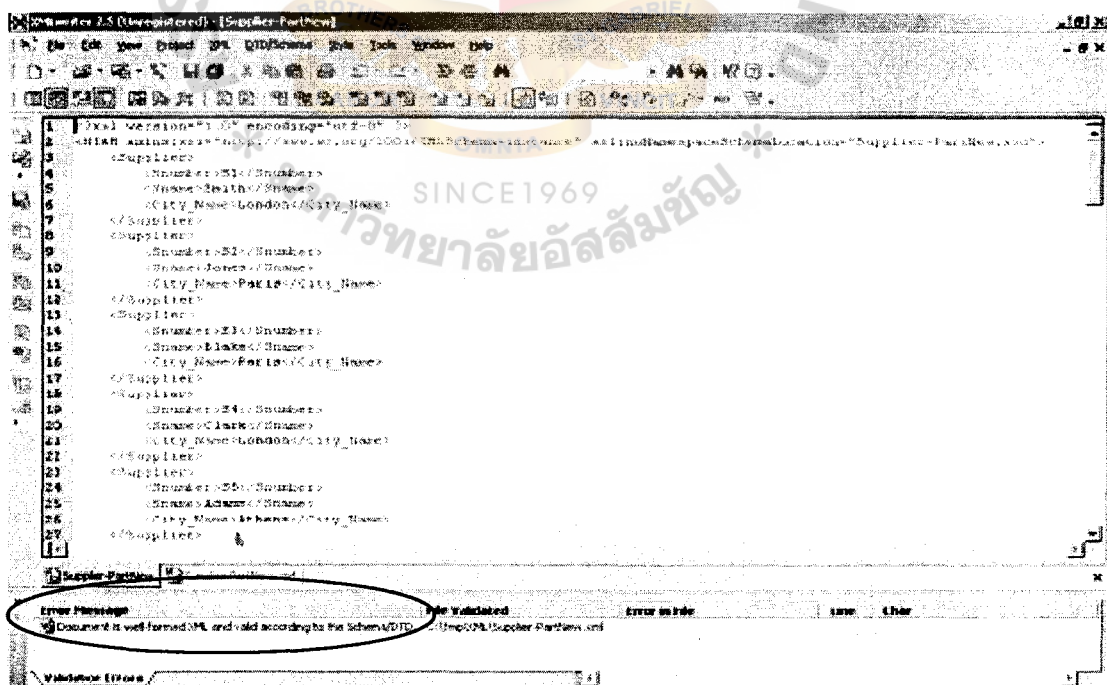


Figure C.6. Checked Validated Supplier-PartNew Xml Document Output.



APPENDIX D
THE META TABLES

<u>ComplexType</u>	NumberOfColumns	PrimaryKey
Supplier	4	Snumber
Part	4	Pnumber
Supplier_Part	3	Snumber+Pnumber

(a) SysComplex Table

<u>ElementCode</u>	ComplexType	ElementOfComplexType	LabelType
1	Supplier	Snumber	Snumber
2	Supplier	Sname	Sname
3	Supplier	Status_Num	Status_Num
4	Supplier	City_Name	City_Name
5	Supplier_Part	Snumber	Snumber
6	Supplier_Part	Pnumber	Pnumber
7	Supplier_Part	No_of_Item	No_of_Item
8	Part	Pnumber	Pnumber
9	Part	Pname	Pname
10	Part	ColorName	ColorName
11	Part	Loc_Num	Loc_Num

(b) SysComplexElement Table

<u>Relation</u>	ParentComplex	ParentKey	ChildComplex	ChildKey
document_mortgagee	mortgagee	name	document	name
document_bank	Bank	name	document	bankname
Document_mortgage	document	documentDate	mortgage	loanNumber

(c) Relational Reference Table

Figure D.1. The Populated Supplier Part Meta Tables.

EntityType	LabelType	ElementType
EntityType1	Snumber	System.String
EntityType2	Sname	System.String
EntityType3	Status_Num	System.String
EntityType4	City_Name	System.String
EntityType5	Pnumber	System.String
EntityType6	No_of_Item	System.Int64
EntityType7	Pname	System.String
EntityType8	ColorName	System.String
EntityType9	Loc_Num	System.String

(d) Object Table

PredicateCode	PredicateName	NoteFD	ComplexGoal
P1	Has	Snumber \leftrightarrow Sname	
P2	Has	Snumber \rightarrow Status_Num	
P3	Has	Snumber \leftrightarrow City_Name	
P4	Has	Sname \rightarrow Status_Num	
P5	Has	Sname \leftrightarrow City_Name	
P6	Has	Status_Num \leftrightarrow City_Name	
P7	Has	No_of_Item \rightarrow Snumber	
P8	Has	Pnumber \rightarrow No_of_Item	
P9	Has	Snumber+Pnumber \rightarrow No_of_Item	
P10	Has	Pnumber \leftrightarrow Pname	
P11	Has	Pnumber \rightarrow ColorName	
P12	Has	Pnumber \rightarrow Loc_Num	
P13	Has	Pname \rightarrow ColorName	
P14	Has	Pname \rightarrow Loc_Num	
P15	Has	Loc_Num \rightarrow ColorName	

(e) Fact Type Table

Figure D.1. The Populated Supplier Part Meta Tables (Continued).

RoleCode	PredicateCode	LabelTypep	Uniqueness	MinCardinality
R1	P1	Snumber	1	1
R2	P1	Sname	1	1
R3	P2	Snumber	1	1
R4	P2	Status_Num	0	0
R5	P3	Snumber	1	1
R6	P3	City_Name	1	1
R7	P4	Sname	1	1
R8	P4	Status_Num	0	0
R9	P5	Sname	1	1
R10	P5	City_Name	1	1
R11	P6	Status_Num	1	1
R12	P6	City_Name	1	1
R13	P7	Snumber	0	0
R14	P7	No_of_Item	1	1
R15	P8	Pnumber	1	1
R16	P8	No_of_Item	0	0
R17	P9	Snumber	2	1
R18	P9	Pnumber	2	1
R19	P9	No_of_Item	0	0
R20	P10	Pnumber	1	1
R21	P10	Pname	1	1
R22	P11	Pnumber	1	1
R23	P11	ColorName	0	0
R24	P12	Pnumber	1	1
R25	P12	Loc_Num	0	0
R26	P13	Pname	1	1
R27	P13	ColorName	0	0
R28	P14	Pname	1	1
R29	P14	Loc_Num	0	0
R30	P15	ColorName	0	0
R31	P15	Loc_Num	1	1

(f) Role Table

Figure D.1. The Populated Supplier Part Meta Tables (Continued).

BIBLIOGRAPHY

English Reference

1. Arenas, Marcelo and Leonid Libkin. "A Normal Form for XML Documents," ACM PODS 2002, June 3-6, 2002, Wisconsin, USA.
2. Arenas, Marcelo and Leonid Libkin. "An Information-Theoretic Approach to Normal Forms for Relational and XML Data," PODS 2003, June 9-12, 2003, San Diego, CA.
3. Becker, Scot A. "Normalization and ORM," Journal of Conceptual Modeling, Issue: 4, August 1998, www.inconcept.com/jcm.
4. Bird, Linda, Andrew Goodchild, and Terry Halpin. "Object Role Modeling and XML-Schema", Proceeding of 19th International Conference on Conceptual Modeling (ER2000), Utah, USA (October 2000):1-14.
5. Bourret, Ronald. "XML Database Products," September 1, 2004. <http://www.rpbourret.com/xml/XMLDatabaseProds.htm>.
6. Chankuang, Narudol and Suphamit Chittayasothorn. "An Object and XML Database Schemas Design Tool," Proceeding of International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2, Las Vegas, Nevada (April 2004): 421-424.
7. Chankuang, Narudol and Suphamit Chittayasothorn. "A Software Tool for Object and XML Schemas Generation," Proceeding of Pacific Rim Conference on Communications, Computers, and Signal Processing (PACRIM'03), Victoria, Canada (August 2003): 675-678.
8. Date, C. J. An Introduction to Database Systems, 7th Edition. NY: Addison Wesley Longman Incorporation, 2000.
9. Daum, Berthold and Udo Merten. System Architecture with XML. CA: Morgan Kaufmann Publishers, 2003.
10. Dennis, Alan, Barbara Haley Wixon, and David Tegarden. Systems Analysis and Design an Object-Oriented Approach with UML, 1st Edition. NY: John Wiley and Sons Incorporation, 2002.
11. Elmasri, Ramez and Shamkant B. Navathe. Fundamentals of Database Systems, 2nd Edition. CA: The Benjamin/Cummings Publishing Company, 1994.
12. Fan, Wenfei and Leonid Libkin. "On XML Integrity Constraints in the Presence of DTDs," PODS 2001, California, USA.
13. Halpin, Terry. Conceptual Schema and Relational Database Design, 2nd Edition. Sydney: Prentice-Hall Incorporation, 1995.

14. Harold, Elliotte Rusty. XML Bible, Gold Edition. NY: Hungry Minds Incorporation, 2001.
15. Hoffer, Jeffrey A., Joey F. George, and Joseph S. Valacich. Modern Systems Analysis and Design, 3th Edition. NJ: Prentice-Hall International Incorporation, 2002.
16. Holzner, Steven. Sams Teach Yourself XML in 21 Days, 3th Edition. Indianapolis: Sams Publishing, 2004.
17. Leung, C. M. R. and G. M. Nijssen. "From a NIAM Conceptual Schema into the Optimal SQL Relational Database Schema," Australian Computer Journal 19, no. 2 (1987): 69-75.
18. Ramakrishnan, Raghu and Johannes Gehrke. Database Management Systems, 3rd Edition. NY: McGraw-Hill Companies, 2003.
19. Routledge, Nicholas, Linda Bird, and Andrew Goodchild. "UML and XML Schema," Proceedings of the Thirteenth Australasian Conference on Database Technologies (ADC2002) Volume 5, Melbourne, Victoria, Australia (January 2002): 157-166.
20. Salim, Flora Dilys, Rosanne Price, Shonali Krishnaswamy, and Maria Indrawan. "UML Documentation Support for XML Schema," Proceeding of International Conference on Information Technology: Coding and Computing (ASWEC'04) Volume 2, Las Vegas, Nevada (April 2004): 421-424.
21. Sceppa, David. Microsoft ADO.NET: core reference, 1st edition. Washington: Microsoft Corporation, 2002.
22. Singer, Michael. "XML Use Almost Doubled In 6 Months Says Survey," May 9, 2001, <http://siliconvalley.internet.com/news/article.php/762281>
23. Suciu, Dan. "On Database Theory and XML," SIGMOD Record 30, no.3 (2001): 39-45.
24. Tittel, Ed., Natanya Pitts, and Frank Boumphrey. XML for Dummies, 3rd Edition. NY: Hungry Minds Incorporation, 2002.
25. Whitten, Jeffery L., Lonnie D. Bentley and Kevin C. Dittman. Systems Analysis and Design Methods, 6th Edition. NY: McGraw-Hill Companies, 2004.
26. Wyke, R. Allen and Andrew Watt. XML Schema Essentials, 1st Edition. Canada: John Wiley and Sons Incorporation, 2002.

Website Reference

1. Fallside, David C. XML Schema Part 0: Primer First Edition, 2 May 2001, <http://www.w3.org/TR/xmlschema-0>.

2. Beech, David, Murray Maloney, Henry S. Thompson, and Noah Mendelsohn. XML Schema Part 1: Structures First Edition, 2 May 2001, <http://www.w3.org/TR/xmlschema-1>.
3. Malhotra, Ashok and Paul V. Biron. XML Schema Part 2: Datatypes First Edition, 2 May 2001, <http://www.w3.org/TR/xmlschema-2>.
4. Walmsley, Priscilla and David C. Fallside. XML Schema Part 0: Primer Second Edition, 28 October 2004, <http://www.w3.org/TR/xmlschema-0/>.
5. Beech, David, Murray Maloney, Henry S. Thompson, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition, 28 October 2004, <http://www.w3.org/TR/xmlschema-1/>.
6. Malhotra, Ashok and Paul V. Biron. XML Schema Part 2: Datatypes Second Edition, 28 October 2004, <http://www.w3.org/TR/xmlschema-2/>.
7. XML Spy, <http://www.xmlspy.com/>
8. XMLwriter, <http://xmlwriter.net/>



St. Gabriel's Library, Au

