DIPLOMAMUNKA

Bara Lehel

Debrecen

2007

Debreceni Egyetem
Informatika Kar

Electronic Invoice Archiving and Reporting System

Témavezető:
Dr. Juhász István
Egyetemi adjunktus

Készítette:
Bara Lehel
Programtervező Matematikus

Debrecen
2007

Ezúton szeretném megköszönni tanáraimnak, elsősorban témavezetőmnek, Dr Juhász István tanár úrnak az egyetemi évek alatt nyújtott segítséget és szakmai támogatást.

Abstract

The usage of the Internet is growing constantly in the whole world, also in Finland. Electronic invoicing is a fast-increasing area in the business world. One of the most widely used electronic invoice format in Finland is TietoEnator's own TEAPPSXML standard, which is a free, XML-based format.

Energiakolmio Oy is using this standard to handle these invoices. Correct and smooth processing of these invoices is essential. Few companies offer value added services for both parties of a transaction; however, one of these companies is the above Energiakolmio Oy. For handling these documents efficiently, the key elements of these invoices must be fast-accessible; they must also support the connection with the company's own information system.

The thesis presents two aspects of archived invoices: one is the archiving of the invoices processed by the employees of this company; the other is a reporting system which is offered for the customers to get up-to-date information about their invoices.

The projects were created in .NET Framework 2.0 using Visual Studio .NET 2005. The programming language was C#, the database server was MS SQL 2000. In addition to the mentioned technologies, it also used XSLT technology.

The archiving component is in production for few weeks, the feedbacks are positive. The reporting service is waiting for the final approve.

CONTENTS

TERMS 1
1 INTRODUCTION 3
1.1 Structure of the Thesis 3
1.2 Energiakolmio Oy 4
1.3 Electronic Invoicing in Finland 5
1.4 Common Electronic Invoicing Methods 6
1.4.1 Scanning of invoices 6
1.4.2 Insular electronic invoice 6
1.4.3 Electronic invoicing using web forms 6
1.4.4 Direct electronic Invoices 7
1.4.5 Electronic invoicing through a consolidator 7
1.5 Benefits of electronic invoicing 7
1.6 Standards 8
2 INVOICE ARCHIVER 12
2.1 Electronic Invoice Sending Process 12
2.2 The Invoicing Application and Architecture 13
2.3 Requirements and Environment 13
2.4 Invoice Archiver 16
2.4.1 Interface for the external applications and libraries 16
2.4.2 XML transformation 17
2.4.3 XML archiving 22
3 INVOICE REPORTING SYSTEM 26
3.1 The Energy Portal 26
3.2 Requirements and Environment 26
3.3 Components 28
3.3.1 Topframe 28
3.3.2 Master report page 33
3.3.3 Archive report 38
3.3.4 Invoicing report 42
4 DATABASE 43

4.1 The Environment....43
4.2 Naming Conventions....43
4.3 The database structure....45
4.4 Stored Procedures....48
5 USED TECHNOLOGIES....52
5.1 NET Framework....52
5.1.1 The C# language....53
5.1.2 ASP.NET....54
5.1.3 ADO.NET....55
6 CONCLUSIONS....57
6.1 Results....57
6.2 Personal experience....58
6.3 Further improvements....58
REFERENCES....60

List of figures

FIGURE 1. Sending process of an invoice....12
FIGURE 2. Use case diagram for Archiving and Reporting System....14
FIGURE 3. The invoices data table....18
FIGURE 4. Flow diagram of an item’s archiving process....24
FIGURE 5. ReportProperties class diagram....31
FIGURE 6. Report header user control....36
FIGURE 7. InfoBox user control....37
FIGURE 8. FacilityInfoHeader user control....37
FIGURE 9. ObjectDataSource and related classes....40
FIGURE 10. The schema of the InvoicingService database....45

List of tables

TABLE 1. Functional requirements for archiver 15
TABLE 2. Functional requirements for the reporting system........................26

TERMS

ADO = ActiveX Data Objects, a set of Component Object Model objects for accessing data sources.

ANSI = American National Standards Institute, it is a private nonprofit organization that oversees the development of voluntary consensus standards for products, services, processes, systems, and personnel in the United States.

CIL = Common Intermediate Language (formally called Microsoft Intermediate Language, all .NET compatible compilers compiles their language into it. The output is called bytecode.

CLR = Common Language Runtime, it is Microsoft's the virtual machine implementation of the Common Language Infrastructure (CLI) standard, which defines an execution environment for program code.

ELECTRONIC INVOICE = Electric invoice is electronic format of common invoice. It helps to automate numerous processes in the buyer's side. It is a modern, reliable, cost-efficient and practically paperless method of handling and processing invoices for goods, services and other expenses.

INVOICE = An invoice or bill is a commercial document issued by a seller to a buyer, indicating the products, quantities and agreed prices for products or services with which the seller has already provided the buyer. An invoice indicates that payment is due from the buyer to the seller, according to the payment terms.

PAYEE ~ **SENDER** ~ **SELLER** = the party in the invoicing process who is selling the goods or services; it is drawing and sends the invoices. From developers aspect payee and sender is the same.

RECEIVER ~ **BUYER** ~ **CONSUMER** = the party in the invoicing process who is

buying the goods or services; it is receiving and paying the invoices. From the developers aspect payee and sender is the same.

SFTP = Secure File Transfer Protocol, it refers to an FTP service over SSH (tunneling a normal FTP session over a secure SSH connection).

SURROGATE KEY = unique identifier in the database for either an entity in the modeled world or an object in the database. The surrogate key is not derived from application data.

VAT = Value Added Tax, is an indirect tax, in that the tax is collected from someone other than the person who actually bears the cost of the tax (namely the seller rather than the consumer).

XML = The Extensible Markup Language is a general-purpose specification for creating custom markup language. It is classified as an extensible language because it allows its users to define their own tags. Its primary purpose is to sharing of structured data across different information systems, particularly via the Internet. It is used both to encode documents and serialize data.

XPath = XML Path Language is an expression language for addressing portions of an XML document, or for computing values (strings, numbers, or boolean values) based on the content of an XML document. It is based on a tree representation of the XML document.

W3C = World Wide Web Consortium is the main international standards organization for the World Wide Web (abbreviated WWW or W3).

1 INTRODUCTION

In Finland information network is part of everyday life, and this development started few years ago. Nowadays almost everything (resolving issues with governmental departments, purchasing flight/ferry/train tickets, home banking) is possible to handle via Internet. The Finnish society has one of the biggest Internet penetration rates in the whole world (more than 60% in 2005 [Internet World Stats]), the vast majority of the Finnish population is familiar with the Internet and more than 50% of them use different services from the Internet on a regular basis.

Using electronic service has many advantages:

- The requested data is up-to-date (in some cases real-time).
- Services are easily customizable.
- It is 24/7/365 uptime.
- It is cheap.

Although the Internet and related services are widely accessible for everybody, this thesis concentrates on enterprise customers, since the implemented solution is targeted for big customers. However, there is no doubt that in the near future there will be a real demand from personal customers as well.

1.1 Structure of the Thesis

The first chapter of the thesis introduces the reader to electronic services focusing on electronic invoicing, standards and benefits on using them.

Chapters two and three present two different applications connected to invoice archiving:

- **Company side solution**: an energy business related company offers invoicing and invoice archiving service towards his customers (in the near future re-

invoicing will be also included in this portfolio). The developed Windows application is used for managing the received invoices, archiving is a part of this applications.

- **Customer side solution**: the company offers a complete reporting system for their clients. As a new field, invoicing, status and history of archived invoices will be included in this reporting system. The created solution is an independent web application embedded into a previously developed energy related portal.

The database schema of these solutions is presented in chapter four.

The fifth chapter will give a short resume of technologies used during developing of these applications.

The last chapter contains the results, personal experiments and further improvements gathered during the planning and developing phase.

1.2 Energiakolmio Oy

Energiakolmio Oy is an independent provider of expert services to parties in the energy market. Beside energy business it offers software solutions in numerous fields, putting focus on energy related services. The writer of this thesis is working in the information technology division of this company (former MVM Energiatieto Oy, which was merged with Energiakolmio Oy in the summer of 2007).

The company employs over 50 people; the IT division has 12 associates.

The services include reporting the electricity sales information and metering data to the market parties, and buying and selling the balance power. In addition, the services include remote reading of electricity meters.

1.3 Electronic Invoicing in Finland

Currently electronic invoicing has a big popularity worldwide. In Finland the suppliers of electronic invoices are increasing, also new solutions in this field are appearing. It is useful to see some basic information about electronic invoices, about the conventional, paper-based invoices and the existing connections between them also.

Invoice has not only commercial value, but it is also an accounting document for both the sender and the receiver. An invoice (paper or e-based) must fulfill the regulations of Finnish Tax Administrations (Verohallinto), more closely the regulations of the VAT laws. The invoice itself is a document made by the seller to document the transaction and it supports accounting and auditing functions also. [Rofhök-Björni 2006, 3]

In Finland a typical invoice includes the following items:

- Date of invoice
- Issuer's VAT-number
- Basic information about issuer and receiver (name, address, etc.)
- The amount and kind of services (or goods)
- Date of delivery or date of payment (in case of prepayment)
- The tax-base for every taxable item, price per unit excluding VAT as well as price and discounts.
- Tax-base
- Amount of VAT to be paid

To be able to send and receive electronic invoices there must be at least an Internet connection, some kind of workstation and last but not least an agreement with an Internet bank (or something similar) where the invoices will arrive. From the sender side the following is required: usually the invoice is sent through the company's invoicing system, but it can also be done through the Internet.

1.4 Common Electronic Invoicing Methods

1.4.1 Scanning of invoices

The most basic method, the originally issued paper invoice is scanned so that the receiver can process it electronically. In case of processing by the receiver or by a third party, usually this method is selected; however a major disadvantage of the method is that the invoices do not use any electronic messaging. Scanning of invoicing is only a small part of the possibilities that electronic invoices can offer for all parties.

1.4.2 Insular electronic invoice

In this case the receiver wishes to receive invoices electronically in her/his system, but the message from the sender (or from an intermediate third party) is not electronic. The advantage of the method is the better internal processing capability of the receiver (at least comparing from the scanned documents). A big, but inevitable disadvantage is that there must be manual interaction of a supervisor (verification).

1.4.3 Electronic invoicing using web forms

A web site, where the invoice details are entered, is a “must have” part of the whole process. In this case usually a third party is present, he is the "physical sender" of the invoices, but not the real one. The real sender is usually a small company. The seller (the real sender) enters the invoicing information on this web form. The request is processed and the created electronic invoice is sent to the receiver. The invoice is in a format that the receiver expects, so its system can process it fully automatically. The invoicing service is run by the request of the buyer (receiver), who expects their customers to create invoices through this system. The service can be run directly by the buyer, or, as mentioned above, by a third party mandated by the buyer. The web site can have additional services: storing, printing and creating electronic copies.

1.4.4 Direct electronic Invoices

The electronic invoice can be collected straight from the invoice provider (who can be the sender). There must be some specific direct connection developed, also one party must implement other parties' invoice standards in case of using different ones in their internal systems.

1.4.5 Electronic invoicing through a consolidator

For a notable amount of companies this option offers a very comfortable option for electronic invoicing: a third party member (bank, application service provider) is involved in the invoice exchange. This type of company will overcome the differences found in different networks, numerous standard of this kind of a company (message formatting, communication protocol) by converting the data between the sender and receiver. The invoices have useful information in both the sender's and receiver's system; even though these parties are using different formats, the involved service will handle the structure transformations. This means that a company can have a single invoicing format for every recipient instead of implementing every single needed format. The consolidator is responsible for the sender's invoice traffic, the format of the invoice and the transfer to the recipient. If the receiver uses a service provider, it is his/her responsibility to collect those electronic invoices and transfer to re receiver.

The current thesis will present an archiving method of invoices. It uses the last mentioned method of collecting the invoices.

1.5 Benefits of electronic invoicing

The benefits of electric documents in general also affect the electronic invoices. Because of the potential benefit of this field, there is a fast growing interest in electronic invoicing. Probably the most obvious and the most important benefit of using electronic invoicing are the reduction of cost and time of processing of the

invoices. The process of receiving a document of this kind is largely automated using reference numbers. The recipient's accounting and payment of the invoices could be automated as well.

The sender's benefit:

- Fast invoicing
- Lowering of costs of material
- Better customer service
- Much less work process (also reduced human error risk)
- Possibility for electronic storage
- Possibility of outsourcing

The receiver's benefit:

- No manual invoice entry
- Automation of VAT services
- Fast circulation of invoices
- Easier filing
- Less errors in data entry and handling
- Automation of accounting
- Reduced cost of invoice handling

Typically, the larger the company, the larger the saving can be, because a large company has more bureaucracy when processing an invoice, so a bigger benefit can automate the bureaucratic processes.

1.6 Standards

There are various standards available for electronic invoicing, the main techniques used in Finland are the following: eInvoice, Finvoice, TEAPPSXML, PostiXml, EDIFACT, etc. The eInvoice was developed by an e-invoicing consortium in Finland, which agreed on technical issues and on the information to be included as well. Two different possibilities are available: XML and ASCII. According to their website,

Finvoice was introduced for common use by the Finnish Bank Association [Suomen Pankkiyhdistys]. TietoEnator has developed its own format (TEAPPSXML), following the main goal: compatibility with both the eInvoice standard and the Finvoice. EDIFACT is used by large corporations. Its importance is decreasing, because the competitors' solutions are more comprehensive and also cheaper.

TEAPPSXML

This thesis presents an electronic archiving solution, where the electronic invoices follow TietoEnators' TEAPPSXML standards. The base of all TEAPPSXML documents is XML. The newest version available in the moment is version 2.7, but the invoicing application followed version 2.5. However, the archiving part of the application could manage documents created following the newest version of the standard. Due to the fact that the received invoices follow the 2.5, it was decided on the supported version. No test was performed with invoices following some other version of the standard.

One of the main elements of the document is the element which represents some kind of value. The normal format of such element is the following:

```
 <element_name>
    <AMOUNT { [SIGN="+" ] | SIGN="-"} VAT="{INCLUDED |
        EXCLUDED}">value</AMOUNT>
</element_name>
```

In case of a positive value the SIGN attribute can be missing, however, its usage is recommended.

Another widely used element type is an element representing a quantity:

```
<element_name>
    <CHARGED { [SIGN="+" ] | SIGN="-"} "Q_UNIT="unit">
        value
    </CHARGED>
</element_name>
```

Like in the above mentioned type, here the usage of SIGN attribute is also optional in case of positive value, but it is also recommended.

Country and language codes follow the ISO 3166 standard codes. They are used for TEAPPSXML COUNTRY-CODE and LANGUAGE-CODE elements. They are not involved in any part of the archiving process, if not taking into consideration the filtering of unused elements at the beginning of processing each invoice.

In case of TEAPPSXML a single electronic invoice document can include a set of invoices. Beside the TEAPPSXML data file the uploaded or downloaded archive file can include attachment files, such as original image of the invoice. In case of the company who is using this invoice handling application the only usable attachment can be a PDF typed image file of the original invoice. The name of the image-file is present in the TEAPPSXML document as well.

The invoice parties present in an invoice are the following:

- **INVOICE_SENDER**: the technical sender of the invoice
- **PAYEE**: actual originator for the invoice
- **SALES_CONTRACT**: sales organization
- **INVOICE_RECIPIENT**: the technical receiver of the invoice
- **RECEIVER**: actual receiver of the invoice
- **DELIVERER**: the deliverer party
- **ORDERER**: the ordering party
- **DELIVERY_PARTY**: the customer to whom the delivery happens
- **PAYER**: the party who actually pays the invoice
- **PAYOR**: the party to whom invoice will be redirected
- **MANUFACTURER**: the manufacturer of the invoices goods or services
- **HOLDER**: the party who holds the delivered goods
- **OTHER_PARTNERS**: any other party

From the above mentioned elements (parties) only the **PAYEE** and the **RECEIVER** elements are mandatory, all others are optional. In fact, in the most cases a lot of parties are the same with the receiver or with the payee.

There is an optional field for posting information. In the case of the present application this information is missing by default, the values are inserted only after receiving the invoice from the sender. The posting service is an optional service for customers, it can be considered like an added value in the customers' service portfolio.

Each invoice contains row level information about the goods (services) the invoice was issued for. Here is detailed information about name, delivery, unit price, quantity, total price of each record. Maybe the most important data for archiving are this information.

The row element has two optional attributes: `ROW_TYPE` and `ROW_ID`. The `ROW_TYPE` attribute indicates the row type. It can be `MAIN`, `SPECIFICATION` or `SUBTOTAL`. `ROW_ID` is used for grouping the rows and it is closely connected to the `ROW_TYPE` attribute. On a normal sales invoice these fields are not needed, in archiving and reporting system they can be named as a critical element.

In this short presentation of TEAPPSXML only some common features of these documents were mentioned and basically only elements and possibilities used in archiving and reporting process. For further information the reader is invited to see the references section.

2 INVOICE ARCHIVER

2.1 Electronic Invoice Sending Process

In the following Figure 1 the phases of the sending process of an invoice are illustrated:

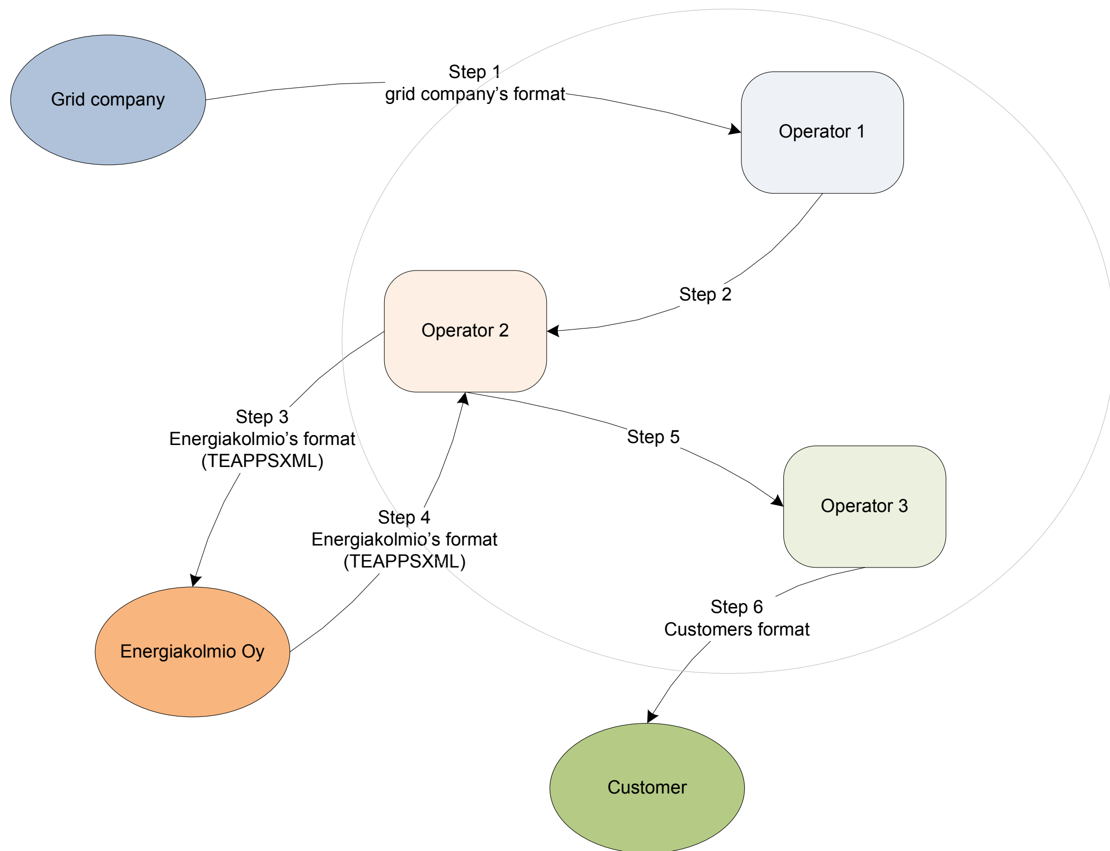

FIGURE 1. Sending process of an invoice

Operator 1 is the consolidator of the *Grid Company*, *Operator 2* is the consolidator of Energiakolmio Oy and *Operator 3* is the consolidator of the *Customer*. They have a contract between each other, where the format of electronic invoice is decided what are using between each others. Two out of three operators or even all three can be

the same. Conversions are made by operators, so the receiver will always get invoices in the required format.

In basic case there are only four parties: the *Grid Company*, the *Customer* and their operators. Energiakolmio offers additional services for their customers (e.g. posting information), so it must be included in the sending flow. The contract is made between *Custome*r and Energiakolmio so that the invoices meant to *Customer* are received by Energiakolmio and after processing forwarded.

2.2 The Invoicing Application and Architecture

The invoicing application is a robust, independently developed application, whereas archiving is a module of this application. With the help of the user interface the user can perform many different tasks which end with a usually mandatory invoice archiving process. The invoices received from the operator can be incomplete or even malformed. In this process the invoices need to be supplemented with some internal information (e.g. client ID used in the internal information system for assuring the connection with existing applications and databases). Additional posting information is not included in the original electronic invoice either. The Windows application gives an interface to the user for entering this information, also to control the whole invoicing procedure.

The application makes a difference between invoicing and re-invoicing; however from the side of the archiving system there is no important difference in handling different types of electronic invoices.

2.3 Requirements and Environment

Before the planning was started, the requirements were gathered that the Invoice Archiver (and the Reporting System) would need to provide.

The following **use cases** were identified for the project. Because of the common actor, the following figure contains use cases for both projects presented in this thesis:

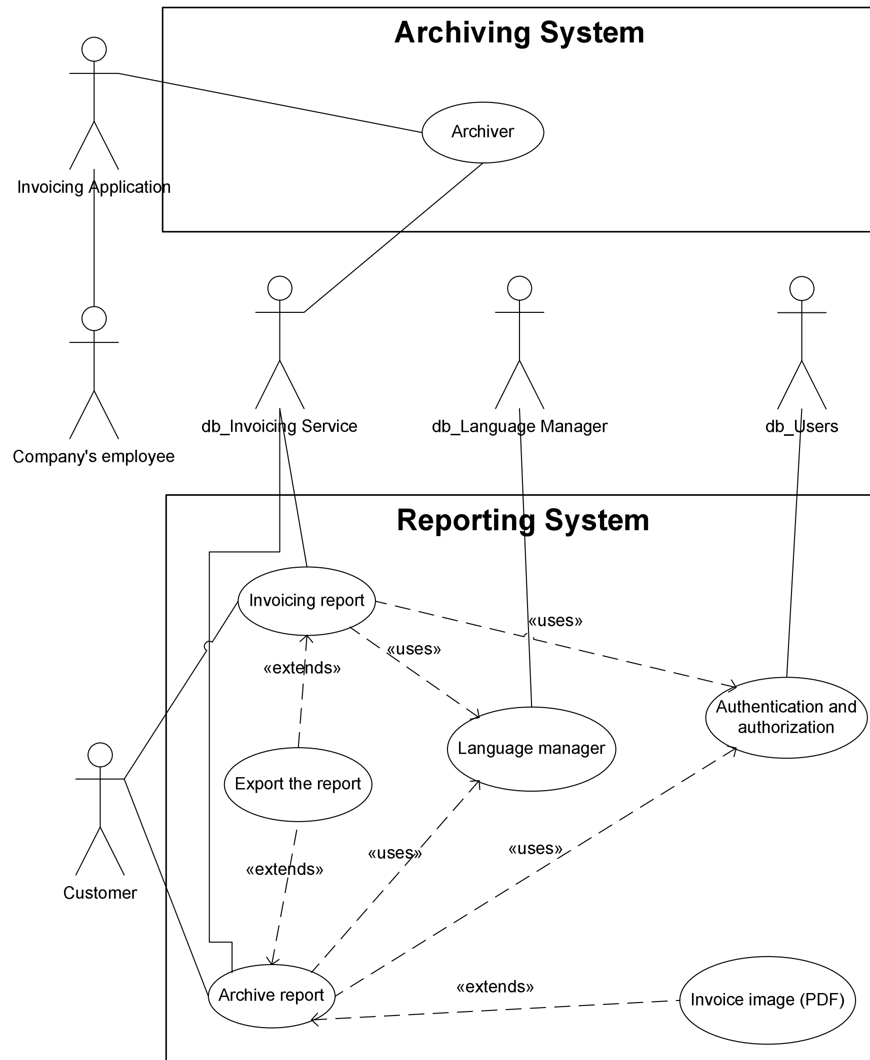

FIGURE 2. Use case diagram for Archiving and Reporting System

You can see the actors which have contact with the system.

Functional Requirements

The following functional requirements were established during the analyzing process:

TABLE 1. Functional requirements for archiver

Identifier	Requirement description	Reference
1	*Archive the invoices* The library has to insert a list of invoices into the database following its structure.	2.4
2	*Error handling* If there is some error, it must handle it.	2.4.1 2.4.3
3	*Notification* A notification must be sent to the caller in the case of some error, especially in the case that the archiving process was error free.	2.4.1

Non-functional requirements and the environment

The application itself was written in C# language under .NET 2.0 Framework, the archiving library as well. The invoicing application must be installed on several workstations. The application upgrades itself from a well defined intranet address if a newer version is available. In the present state it can be used only in the well specified intranet environment since few basic files (mostly configuration and other description files) are deployed on a network drive. Due to pretty easily configurable settings files this behavior is changeable, but it is not recommended. Also the received invoices are stored in a network drive. These invoices are received by a different application in a different process: the Operator of Energiakolmio (consolidator) converts and sends the invoices through a secure tunnel (SFTP). Using a common file space is needed if many users wants to use the application, thus an intranet environment can be the solution for storing these invoices.

The application runs on several workstations, independently of any other machine. Since the configuration settings are stored on a central computer, it affects the behavior of applications.

The selected server is one of the main company's main servers and it is running a Windows Server 2003. For the proper working communication with the database, a server is mandatory. The current database server is MS SQL 2000 Server.

However, the interface of the current version of the application is in Finnish language, in the future there are plans for supporting multiple languages. There are two possible way of implementing this: using local resource files or using the already implemented Language Manager library. The second option will also require permanent database access.

2.4 Invoice Archiver

The invoice archiver library is performing the database archiving of an electronic invoice and its related data. The library has three parts:

1. Interface for the external applications and libraries
2. XML transformation
3. XML archiving

2.4.1 Interface for the external applications and libraries

The interface for the external application provides a calling point to the application for the outside world. It includes a public class, which is the only public class in the project. After creating an instance of this class there are public properties for passing the invoices and also the related data needed for archiving.

The archiving process can be formed in two different modes: simple (single) or batch (multiple) mode. In the case a single is selected, every single item in the invoice list is treated as an individual invoice which has no connection with the others in the list. If archiving of one invoice in the list is successful, the database level commit is called, which means the archiving of the invoice is completed and the changes in the database are definitive. In this mode every single invoice has its own transaction; an error in the middle of the archiving process will affect only the current invoice, after aborting it the archiver tries to process the next invoice on the list.

If the archiving mode is set to batch mode, all invoices are treated in a single transaction in the database level. This option is needed in case the actual

TEAPPSXML document contains multiple invoices. Even though invoices in a single document have some kind of relationship (for example same payee and same receiver), it is not an appropriate way to archive them one by one, not taking to consideration that an erroneous invoice can affect the processing of whole TEAPPSXML document. In this case an invoice, which has some kind of problem or it just does not fulfill the requirements, will invoke a rollback command in database level, so no changes will take place in databases.

The way the caller module will get a notice about the status of the process is the returning value of the archiving method, which is a list of invoices. If the list is empty, it means that every single invoice was archived without any problem and the user can be notified via the graphical interface about the successful termination.

If the returned list is not empty, two cases can be distinguished:

1. In case of single processing mode, this list will contain all invoices not successfully archived, also information about the reason, in details for every single invoice.

2. In the case of batch mode, the list will contain only one invoice, the first one from the original list that did not completely fulfill the requirements.

2.4.2 XML transformation

The goal was to develop a method for XML document archiving without really taking care of its content and without parsing the whole document in the programming level. The way the process was planned: creating a typed dataset in the base of the database schema, using the built-in serialization capability of .NET's **DataSet** objects, somehow transforming the original document in a desirable format for the unmarshalling method (`ReadXML`), performing the unmarshalling and in this way creating an object oriented representation of the invoice. Finally, with the help of some customized **TableAdapter** objects performing the required database operations (using again a built-in feature of .NET framework). About the details

regarding the used techniques please read the subchapter called ADO.NET.

The XML transformation part of the project makes the desired transformations. The technique is **XSLT**, a relatively easy, universal and elegant way of transforming XML documents in different formats. There would be a possibility to load the data without any transformation also, but because some other changes are also necessary, this method was chosen. The output of the transformation is in easier readable format for humans.

The typed dataset was created right after the database structure was finalized. The structure of this **typed DataSet** was generated based on database schema using the proper build-in wizard in Visual Studio 2005. Since the types datasets are derived from the built-in DataSet class, the `ReadXml` (and also the `WriteXml`) method was inherited. The method provides a way to read either data only, or both data and schema into a DataSet from an XML document. In order to be successful, unmarshalling the XML document must fulfill certain criteria. An example how the XML must be formatted in the case of a typed dataset is illustrated in Figure 3:

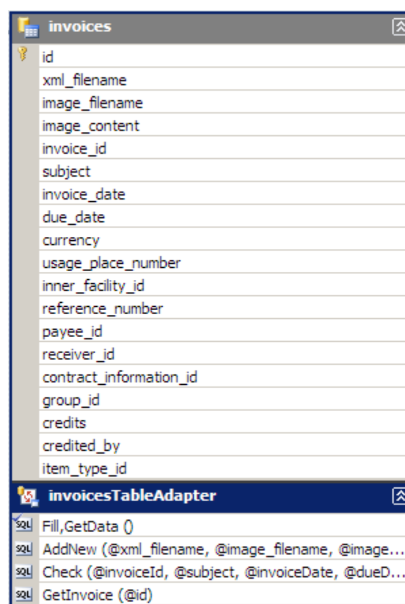

FIGURE 3. The invoices data table

```
<xml xmlns:fn="http://www.w3.org/2005/02/xpath-functions>
```

```
    <Archiver xmlns="http://tempuri.org/Archiver.xsd">
        <invoices>
            <invoice_filename>xxxxxxx</invoice_filename>
            <invoice_id>1234567890</invoice_id>
            <invoice_date>2007-01-
01T0:00:00+00:00</invoice_date>
            <invoice_due_date>2007-01-
20T00:00:00+00:00</invoice_due_date>
            <currency>EUR</currency>
        </invoices>
    </Archiver>
</xml>
```

Right after the creation of the typed dataset, losing some constraints was needed. The reason is that there are some fields in the database, which cannot have `NULL` value, but for few of them it is not possible to determine the value during the XML transformation. A good example is from the table above: in the database the `receiver_id` and the `payee_id` cannot be `NULL`, but in this moment this value is unknown, it is not even known, if the receiver and the sender are already in the database (so they have a well-known id), or they are not yet there (in this case they must be inserted to database, when they will also get an ID).

The XML transformation was made using the XSLT technology and standards.

The transformation was carried out in three steps:

1. **Removing the empty elements**
    - Separating every single logical invoice in the TEAPPSXML document to a stand-alone stream
2. **Simplifying the structure of the original document**
    - Transforming elements into a different path
    - Transforming types in different types
    - Transforming attributes into elements
3. **Creating the proper stream for archiving**

1. Removing the empty elements

Presence of empty elements should not induce big problems during the process. However, there are some issues where it is not possible to pass them easily since they can create problems. The most important issue is that even if it is an empty element (the purpose of the creator was to show the lack of information) few built-in **XPath** functions cannot handle them in the required way (They handle them in a correct and well defined way, just not proper in this case). The most peculiar function is shown in the following example:

```
<xsl:if test="count(CONTROL/IMAGE_CONTROL/IMAGE_FILE) = 1">
<image_filename>
        <xsl:value-of
select="CONTROL/IMAGE_CONTROL/IMAGE_FILE"/>
</image_filename>
</xsl:if>
```

The above code would create a new `image_filename` named element in the output if the number of nodes with path `CONTROL/IMAGE_CONTROL/IMAGE_FILE` is one. The output will produce erroneous output in case of presence of an empty element or if beside a required element there is also an empty element present. However these cases look strange, in the test cases they appeared, so handling it was necessary.

- **Separating every single logical invoice in the TEAPPSXML document to a stand-alone stream.**

One TEAPPSXML document can contain multiple invoices (somehow connected in business level); before archiving separating them is necessary. However, the final version of the Invoicing Application made the separation before archiving process, removing this step was not done, mainly because of the secondary task this transformation does:

2. Simplifying the structure of the original document

- **Transforming elements into a different path**

```
/INVOICE_CENTER/CONTENT_FRAME/INVOICES/INVOICE/PAYEE/CUSTOMER_
INFORMATION/CUSTOMER_NAME
```

element transformed into element

```
/xml/invoice/payee/name
```

Paths of the elements are presented using standard XPath notations.

- **Transforming types in different types**

```
<INVOICE_DATE>
    <DATE>
        <CENTURY>20</CENTURY>
        <DECADE_AND_YEAR>07</DECADE_AND_YEAR>
        <MONTH>02</MONTH>
        <DAY>12</DAY>
    </DATE>
</INVOICE_DATE>
```

to

```
<invoice_date>2007-02-12T00:00:00+00:00</invoice_date>
```

- **Transforming attributes into elements**

```
CONTROL/IMAGE_CONTROL/@SOURCE
```

to

```
xml/invoice/source
```

However, there is a small impact on performance of using an extra transformation, it cannot be considered negligible. Incorporating a second and the third steps can be part of a future version.

3. **Creating the proper stream for archiving**.

In the last step the final transformation is done, where the output must suit the unmarshalling process.

2.4.3 XML archiving

The final part of the process has the role of archiving itself. After unmarshalling the stream, the process will get an object containing the typed dataset with values and some additional information which helps and orients the process.

One challenge of the project was to assure the transaction for the whole process.

Every database table involved in the archiving has a corresponding typed DataTable object in the typed dataset. The values in the rows of these tables are the values to be inserted in the database. Every **DataTable** object contains one DataRow object (`invoce_rowsDataTable` and `invoice_posting_informationsDataTable` are the exceptions). Therefore in case of an error no row must appear in the database. Typed datasets for some reasons are not supporting the transactions natively. The resolution came in a nice feature of .NET 2.0 Framework: support for partial classes.

In a nutshell, partial classes mean that the class definition can be split into multiple physical files. For the compiler the partial classes do not make any difference, during compile time the various partial classes are grouped and treated as a single entity. The biggest benefit is achieved by creating a clean separation of business logic and user interface. In this case creating partial classes for every typed DataTable object can open the way for adding additional features to them.

The following code was created for every table adapter (the example shows the code for `receiversTableAdapter`) :

```
partial class receiversTableAdapter
{
    public SqlTransaction Transaction
    {
        get
        {
            return _adapter.SelectCommand.Transaction;
        }
```

```
        set
        {
            if ( _adapter == null )
            {
                InitAdapter ( );
            }
            Connection = value.Connection;
            _adapter.InsertCommand.Transaction = value;
            _adapter.UpdateCommand.Transaction = value;
            _adapter.DeleteCommand.Transaction = value;

            foreach ( SqlCommand com in _commandCollection )
            {
                com.Transaction = value;
                com.Connection = value.Connection;
            }
        }
    }
}
```

The same transaction (and so the same connection) must be assigned for every table adapter before using them. As mentioned before from a different point of view the scope of a transaction can be one invoice or a set of invoices.

Every table in the database related to the archiving has an auto-increment identity column as primary key. This value is assigned automatically by the database server. Since the value of the identity column is unknown before actual inserting, inserting is performed one by one. After the insertion is done, the identity column value is requested and a new, pending item can be inserted. Inserting and information requesting are not done “natively” by the typed dataset, consequently, these commands were created. The reason is that the built-in `InsertCommand` does not return the indent value, but the number of inserted rows. This reason required for every DataTable, where inserting is performed, the following methods to be added:

```
int? Check ( ... ) {…}
void InsertNew ( ... ) {…}
```

The first method (`Check`) is called twice: first to check if the inserting item is already present in the database or not yet. In the first case it will return with the value of the primary key of the row, otherwise with null. In case of a value the process is broken, the item already in the database, an exception is thrown. If the returning value is null, the `InsertNew` method is called with the proper values. After that the previous method is called again. In this case the returning value will contain the indent value, which can be used for inserting the dependent items. This behavior is presented in the following Figure 4:

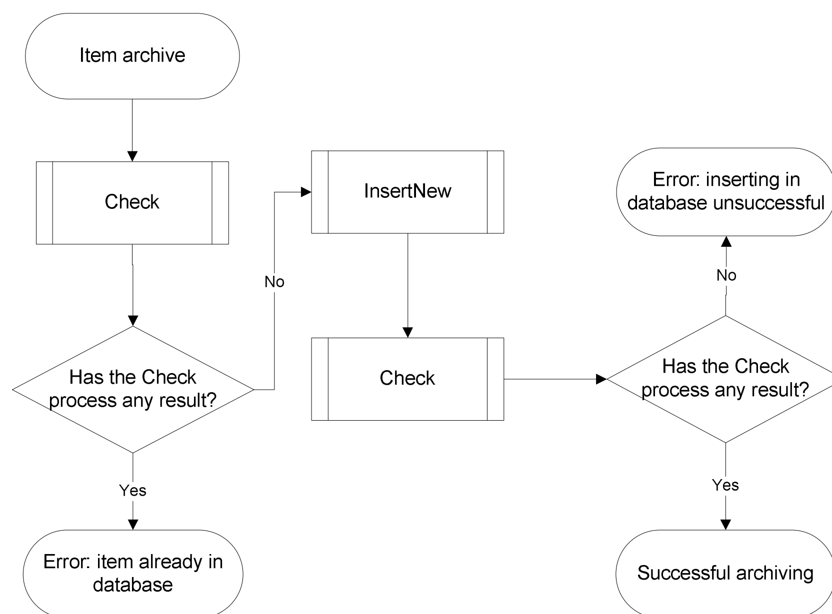

FIGURE 4. Flow diagram of an item's archiving process

The existence of an item is checked by a set of selected columns from a data table, it is not required the equality of all columns to state the existence. A good, but banal example is in the case of `InvoicingService.dbo. invoices` table where it is not needed to check the values of the `invoice_image` column, which contains a byte array representation of the reified invoice (PDF file). The set of fields belonging to unique constraint was defined by the documentation of requirements. Creating the unique constraint in database level it goes without saying, but creating it in code level using the tools of Visual Studio 2005 can be surprisingly backward. The problem is,

that the wizard, and more precisely the query analyzer cannot handle correctly the rows comparison when the value can be null in the database.

Another strange behavior of this IDE appeared during the creation of the `Check` method. The method created at the C# level will always have as returning type object and not the expected (`int?`). A strange solution for the problem is to edit the just created query, and without making any change just finishing the process.

3 INVOICE REPORTING SYSTEM

3.1 The Energy Portal

The Invoice Reporting System is part of a big energy portal run by Energiakolmio Oy. It is called **Enerkey** (www.enerkey.com) and beside the invoice reporting system there is an energy consumption report, cost report, data maintenance, portfolio management and market info application on the portal.

3.2 Requirements and Environment

The use cases analyzed for this project is presented in subchapter 2.3.

Functional requirements

TABLE 2. Functional requirements for the reporting system

Identifier	Requirement description	Reference
1	*Generating Archive Report* The application must be able to generate Archive Report.	3.3.3
2	*Creating Invoicing Report* The application must be able to generate Invoicing Report.	3.3.4
3	*Exporting the report* The application must be able to print, create PDF and generate XLS file from the reports.	3.3.2
4	*Retrieving images* The images of the invoices (PDF) must be returned.	3.3.3
5	*Login* Authorization must be handled.	Explained in 3.2 3.3.3
6	*Language manager* Application's interface must be available in different languages.	Explained in 3.2 3.3.3 4.2

The non-functional requirements for the application

The most important is that it must cog to the design, structure and logic of the host website.

In the current version there are two different reports available for the customers: archive report and invoicing service report. In the near future a third report, called re-invoicing report will be integrated into the system.

The project should handle the following web browsers as clients: Internet Explorer 6.0 and above, Mozilla Firefox 2.0 and above, preferably Opera 8.0 and above. The compatibility with Opera browser was not specified in the requirements, however, many feedbacks found in the Internet mark out, that this browser follows the most the W3C standards, so it was challenging to meet these standards as much as possible.

Due to some limitation of the technologies used on the server side the client code is DHTML (mixture of XHTML, JavaScript 1.1 and CSS 1). Thus the JavaScript should be enabled in the browser.

Because the session is tracked through the browser's cookie, it must also be enabled. The session management and the authentication process are based on a core library used for the whole website.

The portal only uses JavaScript as a client side script language, so no plug-in is required for normal operation. All information is reachable through static pages (text and images), no sound or video is present. Like other part of the portal, this application is also optimized on resolution 1024 x 768 (usable on resolution 800 x 600, only one control (InfoBox) will be not visible).

Like the whole host web site (with some exceptions), the application handles three different languages: Finnish, English and Swedish. The language is set up when the user performs the login on the main page. So the user can select the language, but the not the culture settings. The requirements document specifies that the date-time

formats, currencies and measurements units are displayed in Finnish culture. Numbers and percentage also have their own precision and format.

The whole portal is located on a single server. This is running Windows 2003 Server and IIS 6.0 is functioning as a web server; both 1.1 and 2.0 versions of the Microsoft .NET Framework are installed. This web application is implemented in ASP.NET 2.0, the other parts of the portal were written under ASP and ASP.NET 1.1.

All data is inquired from MS SQL 2000 database server using the Transact SQL language.

The business logic of this web application is simple. However, it is not a stand-alone application, it is embedded in the portal, the logic related to the logged users, the views they can see. The administrative part is all part of a different application.

The reporting part itself consists of selecting some data from the database and creating a view for the user based on the search criteria. Selecting the time period is present for both reports. Sometimes a column is clickable, and clicking on it should generate another request toward the IIS.

3.3 Components

3.3.1 Topframe

The so-called **Topframe** page is the entry point for the user in the Invoice Report web application. The name comes from its position: it is lying on top of the page. An important requirement for it was that the layout of the page and the style of the elements must suit with the already existing reports in the energy portal.

Some functionality is the same as with other top frame applications, hence a common base page was created. However the existing applications are not modified to use this common base page, this was created to ease down possible future developments.

The main elements on the top frame are listed as follows:

- **Facility list**
- **Facility search box, search button and search rollback button**
- **Facility list**
- **Report selector**
- **GO button**
- **Date range selector**

Facility list

The list contains facilities assigned to the logged in user. It is presented using an already created web control (**SmartListBox**) by the company which is derived from **ListBox** control. Beside the inherited attributes an extra functionality is added: after a set of elements is selected, the list is scrolled so that the first visible element in the list is the first selected item.

Facility search box, search button and search rollback button:

Users, in some cases, can have thousands of facilities in the list. These controls help users to apply a filter on the list of facility in a flexible way. In the search box the user can enter search criteria (filter) and the search button applies the filter. Search rollback button appears only if some kind of filter is set. Whitespaces are trimmed from the beginning and end of the entered text.

Report selector

It is a main element of the page. It is a simple **DropDownList** instance, items represent the possible reports. Beside the facility list box, this is the only control which fires a postback event without pressing a submitting control (in this case the facility filter, the facility filter rollback and the so-called GO button): selecting a different report will change the set of controls appearing on this page. At the time of writing the thesis two different reports are implemented (the third one is conceptually planned, but the specification of the requirements is not finalized).

GO button

It generates a server side event for creating the requested report.

Date range selector

This set of controls includes two textboxes, two calendar images and two calendar controls. Calendar controls are bound to images; if the user clicks on the calendar icon, the control pops-up. The selected date is stored in the textbox.

Beside these the following controls are present in **Archive report** option only: text boxes are used for searching by an invoice id, searching by name of the payee and searching in a total value interval. Two **RangeValidator** instance belong to the minimum total value and maximum total value text boxes. RangeValidator instances checks whether the value of an input control is a valid value of the given type (in this case double) and is within a specific range of values. In browsers specified by requirements, the validation is first done on client side and after that in server side also. Browsers not fully supported by this control cannot perform the validation on client side, but server side validation is still available and it is even more accurate. There is a dropdown list for filtering the energy types shown in the report (electricity, district heating, water and all together). It is also possible to choose whether the date range is applied on invoice date (default) or due date and to choose whether the total sum range is to be applied on total neat value of the invoice or on the value without VAT.

The persistency between two requests is assured by **ViewState** technology (seamless) and by **ReportProperties** class. This class stores every parameter needed for a report to be generated. The following UML class diagram presents the ReportProperties class:

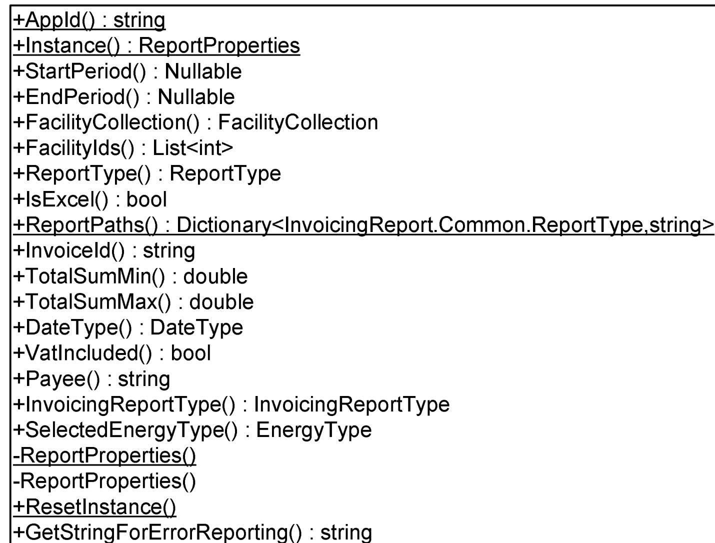

FIGURE 5. ReportProperties class diagram

The class follows the singleton design pattern since only one instance can be instantiated at a time. The instance itself is stored in the session state.

The single object is instantiated when the session is created by the very first request of the topframe page and it is calls the first time the `Instance` static property of the ReportProperties class.

```
public static ReportProperties Instance
{
    get
    {
        if ( System.Web.HttpContext.Current.Session
                [ appid + "reportpropertyinstance" ] == null )
            {
                  System.Web.HttpContext.Current.Session
                  [ appid + "reportpropertyinstance" ] =
                new ReportProperties ( );
               }
         return (ReportProperties)
            System.Web.HttpContext.Current.Session
                [ appid + "reportpropertyinstance" ];
    }
}
```

If the instance is not created yet, the private constructor is called and the instance is stored in the session state.

The `appid` is a constant and it is used as a prefix for keys for objects stored in the session state.

There is a proper field in this class for every control in the topframe. The values of these controls are stored in the fields and when the report generating button (GO button) is pressed, the proper report page is loaded. The report will get the parameters for generating the output from this instance, no `GET` or `POST` parameters are used for this purpose.

The `GetStringForErrorReporting ( )` method creates a readable digest of the content of the instance. It is used for unexpected error logging and email notifications.

One more component is worth mentioning from this web form: **ApplicationLoader**. This control was developed few years ago in this company and it was started to be used for this portal, however, it could be used in any other frame-based sites also. Its duty is to load a page to a well defined frame (in this case to the lower frame). This sounds very stale, but managing this kind of situations can be pretty messy after a while. The control provides full code-behind support for this task without really taking care of the characteristics of HTML language.

This control is used as follows:

1. Registering and declaring the control in the markup file of the web page:

```
<%@ Register Assembly="ET.Web.Controls"
Namespace="ET.Web.Controls" TagPrefix="etweb" %>
[…]
<etweb:ApplicationLoader id="applicationLoader" Target="main"
FormName="enerkey_application" runat="server"
autoregisteronload="False"/>
```

Important is that the control must be placed outside the implicitly created form. The form will create a proper form when required while rendering the page.

2. If no action is required from this control, it must be disabled:

```
applicationLoader.InApplication = true;
applicationLoader.Enabled = false;
```

3. If action is required:

```
applicationLoader.DataFields [ "finalURL" ] = URL
applicationLoader.NavigateURL = "../Loader.aspx";
applicationLoader.Enabled = true;
applicationLoader.Update ( );
```

URL is the address of the requested destination page

Loader.aspx is the page first being loaded in the frame; its duty is then to load the requested page. It was introduced to display a status page while the page is loading in the background instead of showing an empty page without any feedback that the request is under processing.

3.3.2 Master report page

The master page technology is available in .NET Framework since version 2.0. ASP.NET master pages allow creating a consistent layout for the pages in application. A single master page defines the look and feel as well as standard behavior. Individual content pages can be created that contain the content needed to display specifically in that content page. When users request the content pages, they merge with the master page to produce an output that combines the layout of the master page with the content from the content page.

Master page also has a markup file (with `.Master` extension) and one or more code-behind files. In this case `Enerkey.Master` contains the markup, `Enerkey.Master.designer.cs` contains the generated codes and `Enerkey.Master.cs` is available for user code.

The markup page starts with the following directive:

```
<%@ Master Language="C#" AutoEventWireup="true"
    CodeBehind="Enerkey.Master.cs"
    Inherits="InvoicingReport.Reports.Enerkey" %>
```

The working principle and the structure are very similar to normal .aspx files. Exception is that it can contain one or more placeholders.

```
<div class = "datacontainer">
    <asp:ContentPlaceHolder ID="mainpane" runat="server">
    </asp:ContentPlaceHolder>
</div>
```

The placeholder called “mainpane” will contain the specific information for both reports.

The controls placed on master page are reachable from content pages also, but it is not so trivial (**InfoBox** shows additional information for an element on the report):

```
public InfoBox FindControl(string infoBoxId)
{
    return base.FindControl(infoBoxId);
}
```

This method can be called from the content page in the following way:

```
InfoBox masterPageInfoBox =
(InfoBox)this.Page.Master.FindControl("infoBox");
```

It looks quite an uncomfortable method, that's why a public property was created for every control needed to access from content and also a property in the content to access the master page easily. Code placed in the master page:

```
public Control MyMasterpageControl
{
    return this.myMasterpageControl
}
```

And the code placed in the content pages as follows:

```
protected Enerkey master
{
    get
    {
        return (Enerkey) this.Master;
    }
}
```

From now on accessing the InfoBox element is much more convenient:

```
this.master.InfoBox.[…]
```

There was a tricky solution for a problem appearing during the development of the content pages: the built-in **GridView** component cannot render its data source into a table that fulfills the requirements. The main problem was about rendering the header. The layout design required the usage of the `<thead class='header'>…</thead>` html element for the table header for proper printing (in this case if a table cannot fit on a page, on the next page the table will start with the header). The `gridView.UseAccessibleHeader = true;` command will force the GridView's rendering process to include the header row in `<thead></thead>` HTML tag, but it is not possible to set the class of the element. For this problem the following trick was implemented: after sending back the result to

the client, this will run a Javascript block that will make the required changes in the loaded document. The following Javascript code was used:

```
function updateTable ( )
{
    var gridView = document.getElementById ( "<%= TableId %>"
);
    var tBody = gridView.getElementsByTagName ( 'tbody' ) [ 0
];
    var tHead = document.createElement ( 'thead' );
    tHead.className = 'header';
    tHead.appendChild ( tBody.getElementsByTagName( 'tr' ) [ 0
] );
    gridView.appendChild ( tHead );
    gridView.insertBefore ( tHead, tBody );
}
```

Instead of `<%= TableId %>` tag in the response stream there will be included the ID of the generated table. This value is set up in the content pages. The script is run after creating the table on the client side.

Custom controls used on the `Enerkey.master`:

- **ReportHeader**
- **InfoBox**
- **FacilityInfoHeader** and **FacilityInfoFooter**

ReportHeader

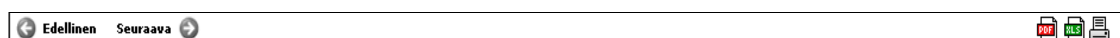

FIGURE 6. Report header user control

Control is placed on the top of the report page, but in the same frame. It has three clickable icons: PDF printing, Excel exporting and normal printing. To comply with the style of the portal, a Next and Previous buttons are simulated by simple images, however since there is no clickable area on these reports, these are useless. The control was used in other project also, thus only few small changes were needed.

InfoBox

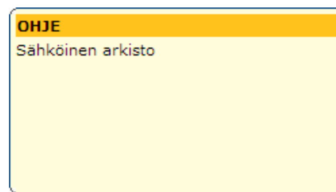

FIGURE 7. InfoBox user control

The **InfoBox** user control contains some kind of quick help about the elements of the page. There is a basic description for every page and also it is possible to assign different text messages for different hotspots. Common hotspots are: table header links, buttons and images. This control was created few years ago for this portal and it was used in this application without any modifications.

FacilityInfoHeader and FacilityInfoFooter

Arkistoraportti (1.11.2007 - 30.11.2007)

16013 19.11.2007

Kohde		Katuosoite	
Yritys		Omistaja -	
Lämmitysmuoto -	Kiinteistötyyppi 15 Toimistorakennukset		

FIGURE 8. FacilityInfoHeader user control

The control provides basic information about the report, the date range and details of the facility included in the report. The figure shows a case when only one facility was selected in the topframe. In case of multiple selected facilities when some of the above mentioned information is not available, a little modified header is rendered. In this case the FacilityInfoFooter (which is located in the bottom of the content page) will contain a list of facility names. This list is not rendered if Excel report is generated.

3.3.3 Archive report

Archive report is a content page requested from the topframe. The code-behind of the page has a previously created class between its ancestor, which handles the authorization and the language related things. The report is generated using the information located in ReportProperties instance. The markup page starts with the following directive:

```
<%@ Page MasterPageFile="~/Reports/Enerkey.Master"
Language="C#"
    AutoEventWireup="true" EnableEventValidation="false"
    CodeBehind="Report.aspx.cs"
    Inherits="InvoicingReport.Reports.Invoicing.Report" %>
```

The difference between conventional ADO.NET page and content page is the reference to the master page. In this case it is “`~/Reports/Enerkey.Master`”. The content itself is placed between `<asp:Content></asp:Content>` server side tags:

```
<asp:Content runat='server' ContentPlaceHolderID='mainpane'>
    <asp:GridView ID="main_gridView" runat="server"
        AllowSorting="True" AutoGenerateColumns="False"
        DataSourceID="ArchiveReportDataSource"
        OnRowDataBound="main_gridView_RowDataBound"
        CssClass="datagrid">
        <Columns>
            <asp:BoundField DataField="InvoiceNumber"
                SortExpression="InvoiceNumber" />
            <asp:BoundField DataField="PayeeName"
```

```
                SortExpression="PayeeName" />
            <asp:BoundField DataField="InvoiceDate"
                SortExpression="InvoiceDate" />
            <asp:BoundField DataField="InvoiceDueDate"
                SortExpression="InvoiceDueDate" />
            <asp:BoundField DataField="TotalEUR"
                SortExpression="TotalEUR" />
            <asp:BoundField DataField="EnergyType"
                SortExpression="EnergyType" />
            <asp:BoundField DataField="InvoiceId"
                SortExpression="InvoiceId" />
            <asp:BoundField DataField="PdfExists"
                SortExpression="PdfExists" />
            <asp:ImageField DataImageUrlField="PdfImageUrl">
            </asp:ImageField>
        </Columns>
    </asp:GridView>
    <asp:ObjectDataSource ID="ArchiveReportDataSource"
        runat="server" SelectMethod="GetDataSet"
    TypeName="InvoicingReport.Reports.Archive.DataSourceFeeder"
>
    </asp:ObjectDataSource>
    <rowclick:EventHandler ID="eventHandler" runat="server" />
</asp:Content>
```

`ContentPlaceHolderID` contains the name of the content pane on the master page, the pane will contain the output from rendering and it is merged with the output of the master page (on the server the merging is done before rendering).

The Grid View component and related classes

The report itself contains a GridView control instance, which is populated with data from an **ObjectDataSource** instance. In this case the sorting by different columns from the header is handled automatically by GridView component, and there is no need to take care of this.The data comes from `GetDataSet ( )` method of the **InvoicingReport.Reports.Archive.DataSourceFeeder** class. The information is not produced directly by this class, it is some kind of proxy class.

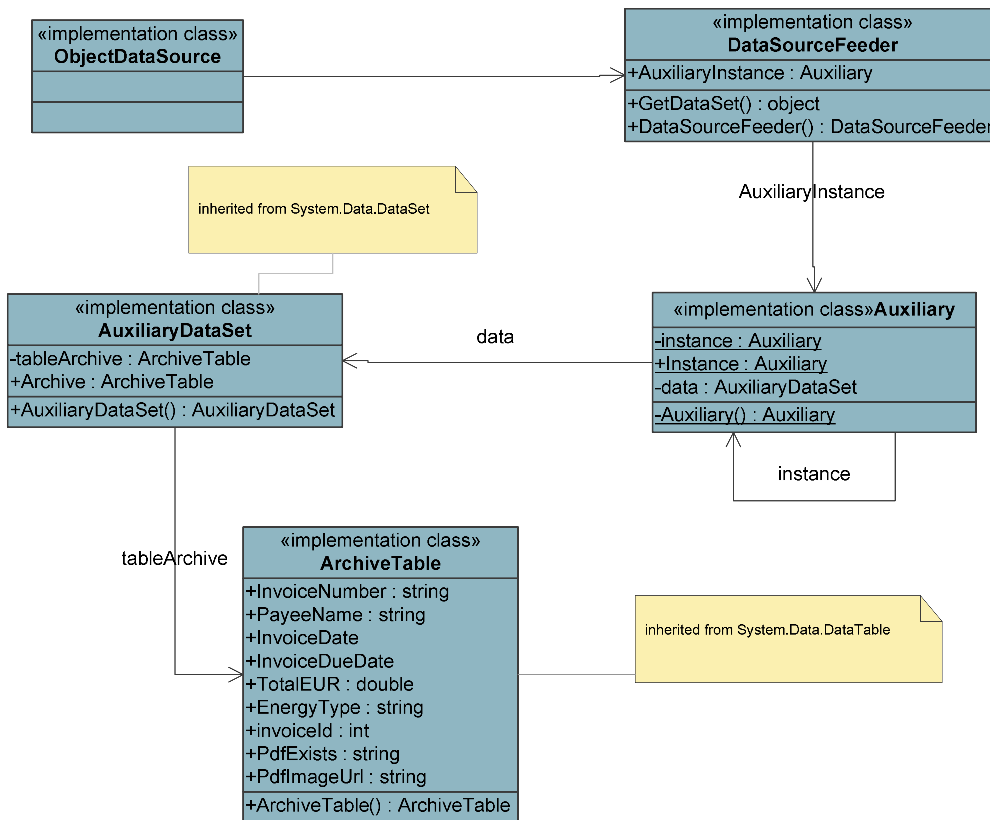

FIGURE 9. ObjectDataSource and related classes

The reason for its usage is: the ObjectDataSource needs a business object with public constructor. The Auxiliary class follows the singleton design patter, it hides its contructor from public. So a proxy class is needed to be created. Its duty is:

- Offering a public constructor
- Offering a public instance memberwise method for getting the required data (`GetDataSet ( )` in this case)
- This method should return the **AuxiliaryDataSet** instance created by the Auxiliary class (this AuxiliaryDataSet instance is created on the first access of the Auxiliary class).

GridView

The GridView component output after rendering is a table. The columns of the table should be placed between `<Columns></Columns>` ASP.NET tags. The **BoundField** class is used by other data-bound controls also (**DetailsView**) to display the value of a field as text. The BoundField object is displayed differently depending

on the data-bound control in which it is used. As mentioned before, in case of the GridView, the control displays this object as a column. In case of DetailsView control the displayed output is a row.

It is possible to customize the output; however the observations show the setting should be done in code-behind. The reason for this is that the Visual Studio 2005 can reformat the markup file and every customization can be lost.

Last two BoundField will not generate HTML output because their visibilities will be disabled after handling the RowDataBound event. The *PdfExist* named BoundField tells to the event handler that the invoice under processing has or has not a PDF file in the database. If yes, a PFD icon is shown in the last column of this row (**ImageField** element) and a client side script is registered on the *OnClick* event. The script contains a Javascript code, which will generate a postback event; it will pass the ID of the invoice to the server, where a custom event is fired. This event handler will return the following response:

```
<script
type='text/javascript'>detailedresults=window.open('../../comm
on/PdfServer.aspx');</script>
```

This HTML code on the client side will open a new browser window or tab and load the **PdfServer** page. The PdfServer will fetch the PDF image from the database and returns to the client.

Persistency layer

The persistency layer can be divided into two parts:

- Auxiliary class
- Company's database access library

The Auxiliary class has the duty of creating the required **SqlCommand** object for data fetching and creating the AuxiliaryDataTable object from the result of executing

the created command. The SQL command contains Transact SQL statements. The parameters are set up using the existing ReportProperties instance. The command is passed to the database access library. The result is filtered and the result typed DataSet is filled with a table and with rows.

The database access library is liable for establishing the connection with the database server, executing the receiver command on the server and returning the result to the caller. The connection is created based on the connection string set by the caller.

3.3.4 Invoicing report

The structure of the report is very similar to the previous report. The differences are the columns present in the report and it does not have any clickable column either. The most important difference is in the persistency layer: the Auxiliary class contains a single database stored procedure call:

```
InvoicingReport.dbo.InvoiceReport
```

4 DATABASE

4.1 The Environment

The Invoice Archiver component of the Invoicing Application and the Invoice Reporting Web Application use the same database. The Windows application inserts (or modifying) data, the web application generates reports based on the data present in the database.

The schema, following the requirements, was created using the administrative tool created by Microsoft for managing databases in Microsoft SQL Server 2000 called SQL Server Enterprise Manager. The target database server was the company's dedicated server for developing tasks. The schema was deployed on the production server using the above mentioned tool's task called "Generate SQL Script".

This thesis covers archiving related features of the Invoice Application Windows application and the Invoicing Reporting System, so the following part of the presentation covers only tables affected by these aspects. However, the database contains other tables also.

4.2 Naming Conventions

The following naming convention was followed during the creation of the tables (tables created not especially for archiving purpose may diverge in this sense):

Table names: contain only small letters, words are separated by an underscore, since it contains instances of an entity, the name refers to the name of the entity in plural form (`invoices`, `invoice_rows`).

Column names: columns are attributes of an entity, that is, columns describe the properties of an entity. So the name of the columns is meaningful and natural, always

refers to the entity, so the name of the table is not present in the name, exceptions are the foreign keys and cases where writing the table name reduces the possible misinterpretations (ex. `xml_filename`, `incoive_date`, `due_date`).

Primary keys: in every table surrogate keys are used as primary keys, they are not derived from any application data in the database. Their names are always `id`.

Foreign keys: The name of the foreign keys has the following structure: [referred_table_name]_`id` (ex. `invoice_id` in table `invoice_rows` or `invoice_postings`).

Indexes, **check constraints** and **default constraints** have the default name generated by the tool.

Due to the fact the applications are mainly used in Finnish environment, the database must support Finnish and Swedish strings. The database and the tables are prepared for handling Scandinavian characters by collations. Collation refers to a set of rules that determine how data is sorted and compared. Character data is sorted using rules that define the correct character sequence, with options for specifying case-sensitivity, accent marks and character width. The used collation for `varchar` typed columns is `Finnish_Swedish_CI_AS`.

4.3 The database structure

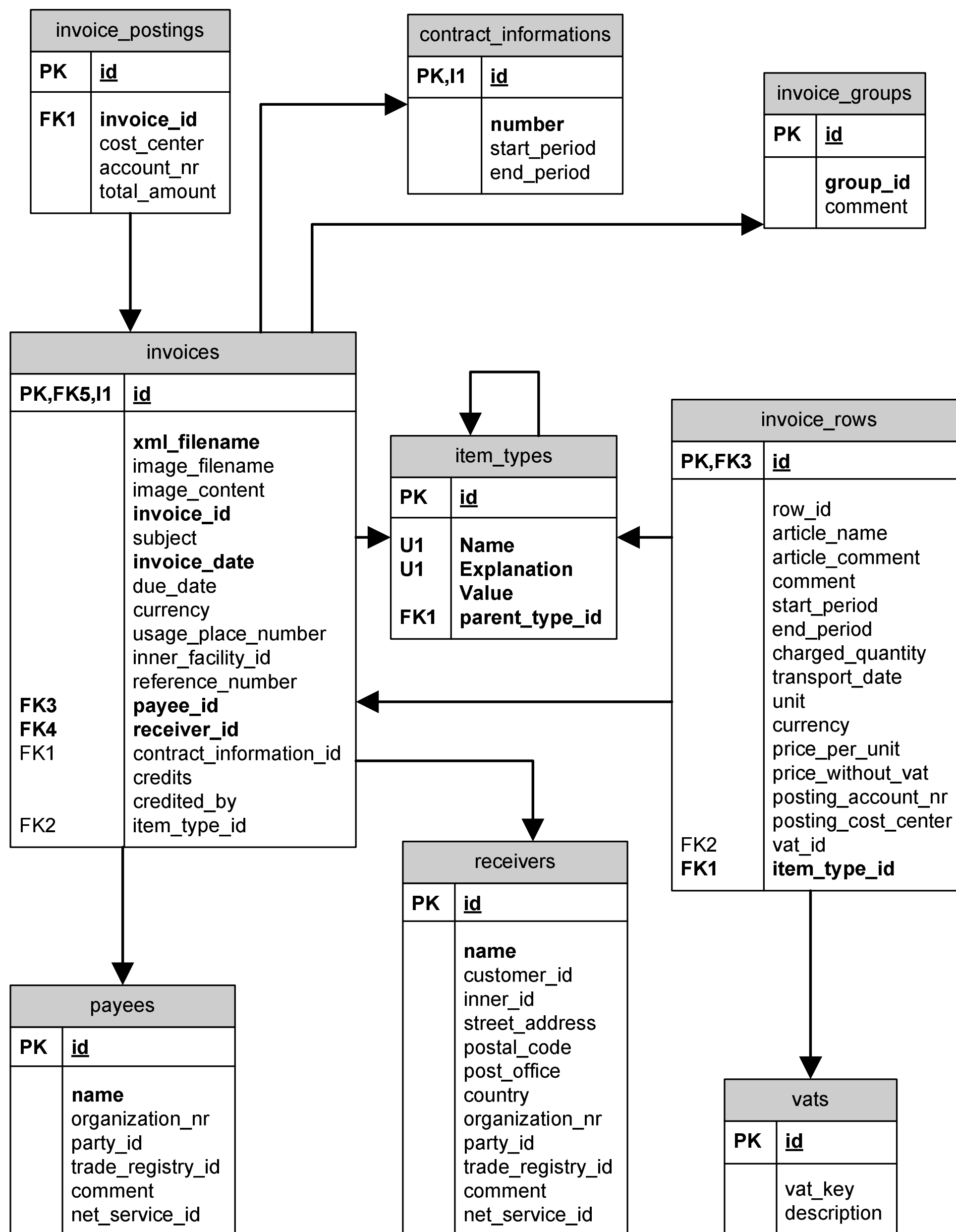

FIGURE 10. The schema of the InvoicingService database

Since the aim was to give self expressive names, only less descriptive fields are explained in some kind of details.

The invoices and the invoice_rows tables are the principles of the whole archiving system.

The xml_filename contains the original filename of the electronic document that includes the invoice itself. This unconventionally cannot be NULL. The image_filename and the image_content store the name of the PDF file and the content in byte array. Because nowadays it is not required to attach the digitalized (scanned) form of an invoice, the image file may not be available. The invoice_id is an identifier used in the payee's information system for identifying the invoice. The invoice_date refers to the issuing date of the invoice, it cannot be absent unlike due_date. The inner_facility_id is one of the most important attributes: it provides one of two available direct connection points with the existing information system of the company. The value refers to a specific row found in a different table of a different database. Since MS SQL 2000 does not support connection realized by foreign keys between different databases, it is impossible to apply foreign keys constraints on this field. The credits and the credited_by columns are not used by these applications. It was not part of the original specification.

The invoice_rows table contains the row elements found in every electronic invoice. The connection with the invoice is assured by invoce_id foreign key. The value of a row element can be calculated using values of price_without_vat and vat_key from vats table. If the vat_id, that makes the connection with the vats table, is null, vat_key value is considered 0. The price of an item is always expressed in neat value. This condition is assured by the Invoicing Application.

The posting_account_nr and posting_cost_center are special fields: they help in receivers' accounting process as it contains automatic paying information. These fields are item row specific. Invoice specific account number and cost center are stored in invoice_postings table. An interesting question is when to apply the row specific and when the invoice specific posting information.

Tables called payees and receivers contain the information regarding to the parties of a transaction: `payees` table contains the seller of the goods, services; `receivers` table in turn contains the buyer, the party whose responsibility is to pay off the value of the purchased items. Tables contain very many similar details; however, the company offers service for the party here referred as receiver, thus more information is gathered for this side. The topmost important field from table `receivers` is `inner_id` since provides the second connection point to the company's existing information system.

In the `contract_information` table is stored few details are stored about the contracts between the parties. The `number` field should identify a valid pact between parties, however the lack of information about how changes in a contract affect this number (is it changed or not) requires the storage to the start and the end of the contract. In current business logic these three values identify a contract.

The `invoice_group` table stores information about the source of the invoice. This is the only information not possible to express in TEAPPSXML format, but used by the archiving system. In the time of deploying to the production server this table had two rows: the first meant invoices created by Energiakolmio's operator (see Figure 1) and the second meant invoices created by EnergiaKolmio Oy. The role of this table appears in the report system, where different kind of reports require invoicees belonging in different groups.

The last table not mentioned before is `item_types`. Both `invoices` and `invoice_rows` contain a field referencing to this table. It describes the type of the service present in the invoice or in an item of the invoice. Field `Value` can have a value < 50, if the description of the item is specific, and >= 50, if the description is more general. The `parent_type_id` field tells the pointing to a generalized description in case of a specific description, or to itself in case of a generalized description. Usually, but not always, rows from `invoice_rows` table points to a specific item description contrary to rows from invoices, which points to a general

item description. Usually, in case an invoice has invoice row items with mutual parent item type, the invoice itself will have this item type.

4.4 Stored Procedures

`InvoicingReport` is the only procedure that should be called from an external application (in this case by the invoicing reporting system). The requirements created many different cases; a lot of these were not possible to generalize so well to put them in a single stored procedure. The main reasons why three different auxiliary stored procedures exist are the mind of creating a code easily readable and easily changed in case of need.

The `InvoicingReport` procedure:

```
CREATE PROCEDURE dbo.InvoicingReport

/*
by cost center = 0
by account = 1
by facility = 2
*/

@groupBy INT,
@facilities ntext = NULL,
@begin DATETIME,
@end DATETIME

AS



IF ( @groupBy = 0 OR @groupBy = 1 )
BEGIN
    CREATE TABLE #ret (
        UsagePlace VARCHAR(100),
        GroupByField VARCHAR(100),
        ElectricityTransferGeneral DECIMAL,
        ElectricitySellingGeneral DECIMAL,
        WaterGeneral DECIMAL,
        DistrictHeatingGeneral DECIMAL,
        ElectricityGeneral DECIMAL,
```

```
        [Sum] DECIMAL
    )

    INSERT #ret EXEC
InvoicingService.dbo.InvoicingReportNotNull
        @begin=@begin,
        @end=@end,
        @facilities=@facilities,
        @groupByAccount=@groupBy

    INSERT #ret EXEC InvoicingService.dbo.InvoicingReportNull
        @begin=@begin,
        @end=@end,
        @facilities=@facilities,
        @groupByAccount=@groupBy

    SELECT * FROM #ret
    DROP TABLE #ret
END
ELSE IF ( @groupBy = 2 )
BEGIN
    EXEC InvoicingService.dbo.InvoicingReportByFacility
        @begin=@begin,
        @end=@end,
        @facilities=@facilities
END
GO
```

The report using this stored procedure can have three different options: result grouped by cost center, grouped by account number and grouped by facility. In case of “grouping by facility” option is selected, the result of `InvoicingReportByFacility` stored procedure is returned to the caller, in other cases the union of the result of `InvoicingReportByNull` and `InvoicingReportByNotNull` is returned. Union of the result is done by the following way:

- Creating a temporary table
- Sequential execution of both auxiliary procedures
- Populating the temporary table with the result of executions
- Returning (selecting) with the content of the temporary table
- Dropping the temporary table

Parameters are:

- `groupBy`: the grouping mode requested by the web application. It can be 0 (grouping by `cost_center`), 1 (grouping by `account_nr`) and 2 (grouping by facility). Grouping by facility basically performs a grouping by `inner_facility_id`.

- Variables named `begin` and `end` define the time range of the report.

- `facilities`: list of facilities present in report. They refer to `inner_facility_id` field of invoices table. Type of this filed is ntext, which means a text with maximum length 2^{11} bytes. It supports UNICODE encoding. The value passed by should be a valid XML text with following structure:

```
<root>[<facilityId value='{number}'/>]…</root>
```

  This value passing method to a parameter is favorable in case of unknown number of parameters. MS SQL 2000 has native support for XML parameters. Using XML parameters has three recommended steps:

  - Creating an internal representation of an XML document by parsing it using Microsoft's internal parser (Msxmlsql.dll). A handler is created and provided to the caller.

```
DECLARE @hFacilities INT

IF @facilities IS NOT NULL
BEGIN
        EXEC sp_xml_preparedocument
            @hFacilities OUTPUT,
            @facilities
END
```

After execution of this code fragment, a tree representation of different type of nodes is created (elements, attributes, texts, comments).

- Using the XML document by the prepared handler:

```
SELECT
        value
FROM
        OPENXML( @hFacilities, '/root/facilityId' )
            WITH ( value int ) )
```

This usage is possible because OPENXML provides a rowset view over an XML document. On a rowset it is possible to use Transact-SQL statements.

- Removing the XML handler. This step is not mandatory but highly recommended.

```
IF @hFacilities IS NOT NULL
BEGIN
        EXEC sp_xml_removedocument
            @hFacilities
END
```

Skipping this phase can call forth a serious impact of database system performance because the system will not automatically free up the memory. The MSXML parser can use a maximum of one-eighth of the memory available for the MS SQL server.

5 USED TECHNOLOGIES

The following few pages present technologies used while the applications were created. There were not so much options when choosing the programming environment since the company is specialized on programming in .NET environment. These are one of the first projects using the version 2.0 of the framework. As the programming language, C# was the obvious choice. Almost every running project is developed in this. My opinion is that this language offers every possibilities of the .NET framework in a very convenient and implicit way. It also has a full compatibility with the intermediate language.

5.1 NET Framework

Microsoft.NET is a general-purpose software development platform, similar to Java. Microsoft's official definition about this framework is the following:

> The .NET Framework is a development and execution environment that allows different programming languages & libraries to work together seamlessly to create Windows-based applications that are easier to build, manage, deploy, and integrate with other networked systems. [MSDN]

It is included in Windows operating systems from Windows Server 2003, but it can be installed for the older version also. In the moment of writing of the thesis there are four final versions on the market (1.0, 1.1, 2.0 and 3.0). Since the applications presented above were written under .NET 2.0, this subchapter will focus on features appearing in this particular version.

The .NET Framework 2.0 was released with Visual Studio.NET 2005, Microsoft SQL Server 2005 and BizTalk 2006 products. Very important improvement came with MS SQL Server 2005, the database system with CLR hosting. This offers for developers

the possibility of building stored procedures, triggers and even data types in any of the .NET-compliant languages, such as C#.

In this version the *generics* debuted, which enables creating a strongly typed collection, providing fewer chances for error in runtime, also giving Intellisense feature. Generic collections with a type will directly store elements with their own type, casting to object during storing and casting to their own type after accessing can be skipped, so the performance is increased. With the appearance of generics it is possible to create Nullable types (using `System.Nullable<T>`). This is ideal for expressing the case, when the variable has no value. It comes with a new type modifier (ex. `int?` result).

Anonymous methods enable the developer to put programming steps within a delegate that can be later executed instead of creating an entirely new method.

5.1.1 The C# language

Microsoft offers an alternative for the developers, which do not want to struggle with the legacy of older languages inherited into VB.NET and C++.NET (e.g. case insensitivity in VB.NET). They created C#, a completely new language designed specifically for .NET platform what did not have to inherit old language elements and structures for maintaining the backward compatibility. Although Microsoft describe the language as a modern, simple, object-oriented and type-safe programming language derived from C and C++, independent developers would also add Java as a source of ideas of the developers of the language. The language supports: consistent set of basic types, built-in support for automatic generation of XML documentation, automatic clean-up of the allocated memory, full access to the .NET base library as well as access to the Windows API, pointers and direct memory access available if required, support of properties in the style of VB, support for creating ASP.NET web pages.

5.1.2 ASP.NET

ASP.NET is a base component of .NET Framework and it is called the successor of Microsoft's ASP technology. Using this web application framework, developers can build dynamic web sites, web applications and XML-based web services. Programmers are encouraged to develop an application using event-driven GUI paradigm rather than conventional web-scripting [Esposito 2006, 7]. The framework tries to combine the existing technologies like the inherited stateless web environment, JScript (Microsoft implementation of Javascript) and internal ViewState component into a persistency enabled environment [Robinson 2002, 753-755]. Significant differences between ASP and ASP.Net:

- compiled code assures less error in runtime and also faster run
- allows programming web sites in .NET compatible languages
- separated business logic and presentation layer
- session state can be persistently stored in an SQL server, which provides that two separate processes in the same machine or on a different machine can access it

Version 2.0 brought big improvement in standards compliance as well as the browser detection feature. All controls generate valid HTML 4.0, XHTML 1.0 (default) or XHTML 1.1 output.

The Page class

Every web page (.aspx and .aspx.[cs/vb/cpp] created in ASP.NET is a class derived from the Page class.

It is an interesting question, how this is possible for .aspx files since they contain only HTML and ASP.NET markup. The answer resides in the fact that before rendering the runtime a source file is generated from the .aspx file. This class is derived from .asp.[cs/vb/cpp] and since this has Page class as ancestor, the result file will also have this file as ancestor.

5.1.3 ADO.NET

The .NET Frameworks ADO.NET called component provides easy data access for applications. It is called the successor of ADO as well as ASP.NET is the successor of the ASP technology, but its concept was also redesign. Since in applications presented in this paper no direct database access was implemented (accessing itself was performed by a previously created component), only the often used classes are presented.

DataSet: it is basically an in-memory cache of data retrieved from a data source (e.g. database). It is made up of a collection of tables, relationships and constraints.

DataTable: represents one table of in-memory relational data. The data can be populated from numerous data sources. From MS SQL Server 2000 the evident object for data retrieval is the DataAdapter class.

The structure (schema) of the table is represented by the columns and the constraints. Definition of the columns is made using **DataColumn** objects; the constraints are made by **ForeignKeyConstraint** and **UniqueConstraint**. Columns can be expression columns as well. The rows are **DataRow** objects; they contain the actual data in the table. Every DataTable object has a collection of DataRow objects (Rows), also of DataColumn and of constraint objects (Columns).

The DataSet class is also tightly integrated with XML: it is easy to save or load XML data from or into a DataSet instance. XML usage creates data transfer independent of the programming environment used, thus it can be used for interoperability between different technologies or environments.

Creating typed DataSet (sometimes called strongly typed dataset) is a special feature present in Visual Studio .NET developing environments. The tool generates an XSD file based on a data source or user interaction. Another tool is created from this XML based XSD file a set of classes. These classes inherit the DataSet, DataColumn and DataRow classes. In case of an external data source, an appropriate **TableAdapter**

class is also created for every DataTable. Creating of XSD files are also possible by hand.

Main advantage is the build-time type safety, since the type of the columns in the rows are not simple objects anymore (of course it can be set explicitly), they have specific types. When using a data set to assign value to instance members in business objects, the data set type must match the business object member or a conversion from the type is needed. It also enables the Visual Studios *Intellisense* feature.

The reader needs to pay attention to the following remark: only basic built-in types are supported for DataColumn type in this way.

6 CONCLUSIONS

The goals proposed in the requirements were accomplished in both projects:

- The Invoicing Application is already released for internal usage and the feedbacks are promising.
- The Invoice Reporting System in the moment of writing the thesis is waiting the releasing approve of the procurer party.

However, I have no doubt there are many possible fields for further development.

6.1 Results

Looking back on the goals and on the whole designing and develop process, my opinion is that both projects were well designed and the resulting library and application are usable and well constructed.

The archiver is working properly; the incoming electronic invoices after manual processing are going to the database. In case of an error a proper and detailed error message appears. As far as I know, since production date the archiver threw only one error: after inspecting the situation we found that the invoice arrived from our operator did not follow the TEAPPSXML standard.

In the development phase few parts of the plan were changed, but these things did not affect the result library. Changes were needed after founding few limitations of the typed DataSets and odd behavior of Visual Studio .NET 2.0.

There is no feedback yet for the reporting system, but in my opinion the objectives are achieved. Following the initial plans was easier for this project. There are already some visions about future requests, but these will come in a future version of this product.

6.2 Personal experience

Since this was my first project under .NET Framework version 2.0 I tried to use the new features appeared in this version. Using the typed DataSet was very challenging. Its usage is very comfortable and smooth, but creating it is far from being tranquil as I personally thought.

Also the way I planned the TEAPPSXML stream reading was quite interesting for me. I would not say funny since there were few moments I thought it is not possible to finish in the way I wanted. The XSLT transformation was the top most challenge; unfortunately the documentation I found about this topic was not really relevant for cases I encountered.

An interesting aspect was working with another team, who created the specification. It became even more fascinating during creation of the reports whilst the requirements were changing. Choosing the iterative development method was quite a good idea in this case.

6.3 Further improvements

As I mentioned in the thesis there are numerous things I could improve. Watching back to the work I observed that few things I would do differently as I have done before. One of my biggest regrets is about dividing the XML transformation part into three different steps. The reason I could not decide in a different way was the fact that the initial requirement for the archiving library was the ability of transforming (not archiving) multiple invoices at a time. However, the change in Windows application regarding to invoice processing method made this feature useless. It is unlikely that the structure of this application will radically change in this aspect, so optimization should be done.

The reporting system will include a new report type in the close future. Blissfully the base of this web application was planned in the way that it supports future reports without major changes in the existing code.

REFERENCES

Esposito, D, 2006. Programming Microsoft ASP.NET 2.0. Redmond: Microsoft Press.

Robinson, A., Allen, K.S., Cornes, O., Glynn, J., Greenvoss, Z., Harvey, B., Nagel, C., Skinner, M., Watson, K., 2002. Professional C#, 2nd Edition, Birmingham: WROX Press.

Rofhök-Björni, A. 2006. Electronic Invoicing in Finland - attitudes towards electronic invoicing by financial managers in small- to mid-sized companies. Swedish School of Economics and Business Administration, Master's of Science Degree Programme in Accounting.

Internet World Stats, referenced December 2007, http://www.internetworldstats.com/top25.htm

MSDN (Microsoft Developer Network), referenced December 2007, http://msdn2.microsoft.com/en-us/netframework/Aa569294.aspx

Suomen Pankkiyhdistys (The Finnish Banker's Association), referenced December 2007, http://www.pankkiyhdistys.fi/finvoice/