The original concept of object classification system

(Oryginalna koncepcja systemu klasyfikacji obiektów)

mgr inż. MONIKA BISKUPSKA, mgr WIOLETA BORYSEWICZ, mgr GRZEGORZ MAZURKIEWICZ

Instytut Maszyn Matematycznych, Warszawa

In recent years the growing number of paper documents as well as the digital content exceeds the capacity of manual control, especially in SMEs sector. Employers and employees are increasingly required to handle wide ranges of information from multiple sources. Among the economic actors SMEs are those who are especially at risk of information growth as accessing, storing and managing relevant information are becoming increasingly difficult tasks with their limited resources.

As effective document management is the key to the effective implementation of knowledge management [1] the main idea of the presented software was to design such a document classification system which could be simple for users and yet as effective as possible. It is especially important in SME sector where time and simplicity in the application of the new solutions are essential. As the open API solution has been applied in this software it will allow for future integration into larger systems, i.a. knowledge management systems, which is in line with future market trends [2].

This paper first outlines the features of the original idea of document management system with its strengths, weaknesses and problems which occurred with time and continuous development of IT technologies. Next chapter presents brand new solution with its *Groups* concept as well as automatic labelling and naming patterns and the *Notes Framework*. In the following section flexible user interface solution is described with its user-friendly drag-and-drop editor. The paper concludes with summary and further considerations.

Original idea

The software was designed and built with small and medium enterprises in mind and its goal was to mimic the paper document management system such companies already had. It was meant to attract users who are not experienced with computers and often do not find it necessary to use a document management software at all. The idea was to apply the already known workflow to another medium and show users its capabilities.

As such, the first iteration of the software was a simple database of document metadata used by most of target users. It stored well-known data, such as:

- default information about a document – like name, identifier etc.;
- description – a short information about the content of the document;
- various types of dates – like registration date, delivery date or expiration date;
- persons assigned to the document – like registrant, deliverer, receiver etc.;
- attached file – the original source of the stored document (with full-text-search capabilities [3]);
- various information about document location and classification – like a subject, a case or a binder;
- different serial numbers – informing about the order of the document in the relevant software structures, i.e. a location of the document in a binder.

The software storing that metadata of the documents worked perfectly for small and medium enterprises with standard needs as it imitated typical workflow of the documents in such circumstances. But if the company wanted to extend the original structure, they were met with many difficulties. It was necessary to order, design and deploy a special module which extended the original product. All that was a time-consuming task and could lead to disorganize the standard enterprise's workflow.

Moreover, the user interface containing various metadata forms were carefully implemented in design time during implementation phase. Therefore, any changes to the default layout were not allowed during the software's runtime. As there were many users' requests for enabling modification of the forms' layout, which had to be designed and then deployed on the client machines, in a new built from scratch version of the software the idea of a layout configurator was implemented.

… and its extensions

The software was designed to be universal and fit the requirements of most of potential companies. But as the client arrived with his specified needs, it was shown that the software did not support all of them. This resulted in the need of changes. Therefore, there have been designed many modules to fit the needs of some of the small and medium enterprises, for example:

- module "complaints and requests" – designed and implemented to enable to service complaints and requests of a company. It entailed a redesigned database structure for storing information about a complaint or request application, result of the settlement, the actual settlement date, place and care of the unit settling the case etc. Moreover, it resulted in the need of modify the user interface;
- module enabling registration of the contracts – designed especially for a hospital using the software. It allowed storing information about the type of the contract, gross or net amount for which the contract was signed and some important dates of a contract lifetime. As the above, this module entailed a redesigned database structure and user interface;
- classification structures gained their own metadata, like "start date" and "due date" and "in progress", "closed" and "delayed" states for "cases" [4].

This listing describes only a few modules that extended the original software. But it can be imagined that all of them forced not only to redesign the database structure, but also to implement new forms supporting this changes. This in turn made necessary to rebuild the software and finally update or even reinstall it on the client machines. Not to mention the large number of the installers that had the ability to install one or more particular modules of the software.

Furthermore, the extensions of the original software had immanent problems with the backward compatibility. When the module was installed to the original software, it was difficult or even

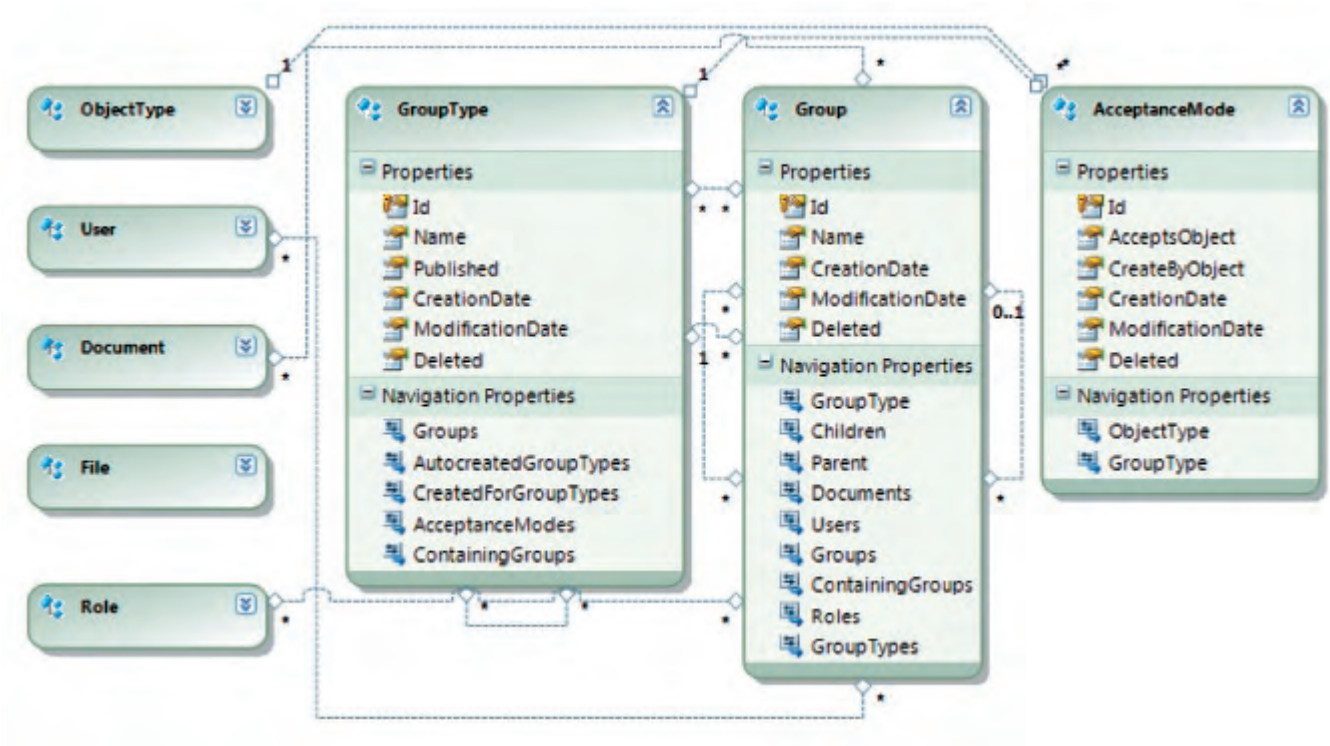

Fig. 1. The Entity Framework diagram for parts of *Groups* concept. Rys. 1. Część diagramu Entity Framework dotycząca koncepcji Grup

impossible to undone those changes. Also, this resulted in an issue with lateral compatibility as the updates to the current version of the software with many modules could be problematic. In fact, the updates had to be prepared separately for each version containing a certain module.

Moreover, the structure of the software's code described in this paper was very hard to maintain. As there was the core version of the software, its many extensions and the installers for every possible configuration, the developer team had to perform a large amount of work to update the code and prepare a suitable version of the updates for the particular enterprises [5].

All these issues led the team to the necessary decision of designing and implementing a new version of the software from scratch.

New solution

The cutting-age technology from a few years before sometimes instead of a matured solution ends up as a dead end with no updates at all or updates consisting only of breaking changes. The document management software in question was very hard to maintain. When compatibility with current technology trends (like modern operating systems) started to become an issue, the decision of a major update was made. This update was an opportunity to redesign some core mechanisms of the software to encompass all the changes made to the classification module as one consistent, streamlined component easy to modify and upgrade later.

Groups concept

The first issue to target was the "cases" and "binders" concepts. They were added after initial release and their integration into original data structures were less than seamless. Moreover, there were some end users who requested different categorisations, like "contracts" or "complaints and requests", to group their documents and use some additional functionality.

In the new edition all those categories are generalised into a new object type called *GroupType*. One such type is a "binder", another type is "case" and the third one is "contract". New types can be added in runtime if necessary. Instances of such types are *Groups* like "binder one", "case complaint no. 1234" or "contract with Good Company Ltd.".

A *GroupType* defines what object types can be added to a group of that type and in what quantities. There are five different possibilities of object type acceptance:

- ✓ *None* – groups of this type don't accept such objects at all.
- ✓ *Many-to-many* – groups of this type accept unlimited objects of this type and objects of this type can be placed in any number of groups of this type.
- ✓ *Many-to-one* – groups of this type accept unlimited objects of this type, but any object of this type can be placed in only one group of this type. This is particularly useful in dividing objects into exclusive partitions, like "customers" and "employees" for *Persons*.
- ✓ *One-to-many* – groups of this type can contain only one object of this type, but that object can be placed in any number of groups of this type. For example, it can be used to indicate the only one *Person* responsible for a group.
- ✓ *One-to-one* – groups of this type can contain only one object of this type and this object can be placed in only one group of this type. It is crucial functionality for automatically labelling objects according to a naming pattern.

Groups can contain different object types – documents, people, other groups or roles – and have their own metadata forms containing various description and logic fields. Thanks to the *GroupType* definitions and the *Notes Framework* (which will be discussed later), *Groups* can completely replace not only existing in the previous version "cases" and "binders", but the requested "contracts" and many other, not yet clearly defined organising structures.

Automatic labelling and naming patterns

Labelling, tagging, numbering and otherwise describing the documents in a unique way was a very important part of the original software and end users often requested new kinds of automatic

and manual identifiers for their purposes. In the new version the labelling process was integrated with *Groups* concept via the previously mentioned *One-to-one* acceptance mode. Groups taking part in this process were named *OtoGroups*.

Those *OtoGroups* are created automatically when needed. They contain the object which they identify and sometimes some other objects like a parent group. The identifier itself is carried in the name of the group. That name is generated using the *NamingPattern* entity, a configurable utility attached to the *GroupType* of the *OtoGroup*. Using the *NamingPattern* it is possible to achieve most if not all of the user-demanded identifiers, like:

- global number of the object, based on the total number of currently present objects of this type in the system, taking into account objects previously removed;
- number of the object local to the particular group, for example continuous numbering of documents in a "binder" group.

Numbering can be restarted according to a few predefined circumstances like with a specific date, formatted or even changed into lettering, and of course all those numbers can be concatenated with user strings to create document identifiers like "175/2012/52-c" (where "175" is a global number of all documents processed in this year, "2012" is this year, "52" is a name of a particular "binder", "c" is a current number of the document inside this "binder" and all of the backslashes and dashes are user entered strings). The automatically generated label can be changed if needed and recreated in case of a user's mistake.

The collection of usable identifier parts is to be further expanded with data collected from fields originating from the *Notes Framework*. Also, the logic inside the particular counter is to be improved with capabilities of simple mathematical operations to provide options to start from the arbitrary number or to skip every certain number.

Notes framework

In the original software were a lot of "dates" for various purposes, like "delivery date", "due date" or "expiration date". Most of them described the document, but some were parts of a "case" or a "binder" metadata. In the updated edition, all this "dates" were collapsed into a single entity called *NoteDate*.

NoteDate can be seen as a definition of a label that can be attached to another object. It describes the metadata of a date value – defines types of objects this date refers to, if the date is mandatory for those objects, if the date is editable after setting it once, date boundaries, default value and the name for this date. Such a configured *NoteDate* is then used to display a value field on the metadata form of the object. Values are stored in an object called *NoteDateInstance*.

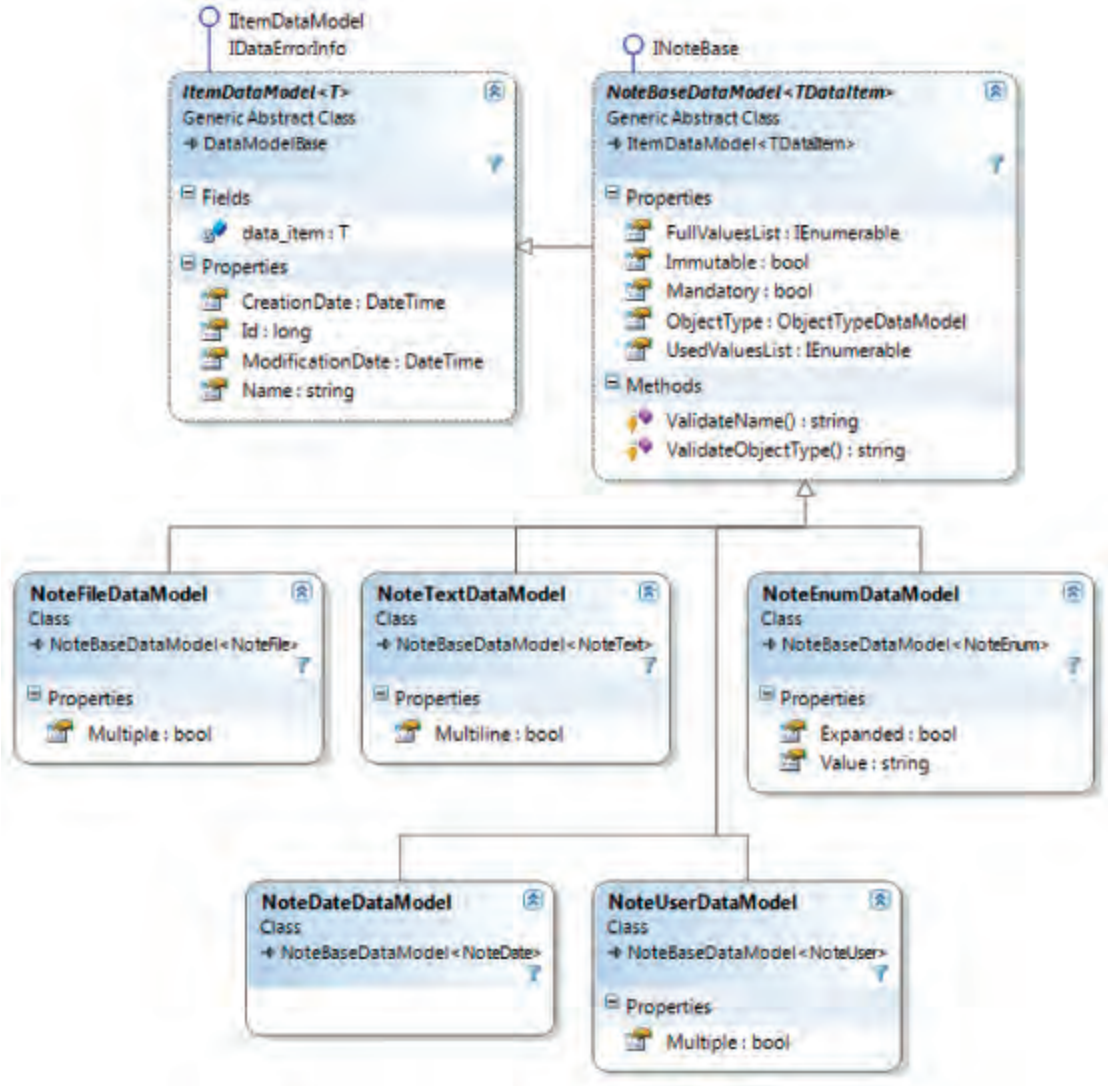

Fig. 2. *Note Framework* partial class diagram. Rys. 2. Część diagramu klas Notatek

This initial concept was then widened with the introduction of *NoteText* and *NoteEnum* entities and then expanded into the whole *Notes Framework*. Its idea is to replace the rigid, haphazard fields of metadata form by the reusable, flexible system than can be configured during runtime. While complicated for end users to set up by themselves, the framework is much easier to maintain and deploy in different configurations than the old approach requiring recompilation after minute changes for particular users. The configuration phase is done once within the installation scripts, and the helpdesk staff can reconfigure it in runtime should the need arise.

On the current stage of development, the *Notes Framework* consists of:

- *NoteDate* – replacing various "dates" in the original solution, in particular "delivery date", "due date", "entry date" and "expiration date". It is also used for system dates like "creation date" and "modification date" on every object. Dates can have some default value (like "today") and some boundaries (like "after 2010").
- *NoteEnum* – replacing binary flags from the original solution and expanding this concept to multi-valued lists. The enumerations can be single- or multiple-choice, represented by radio-buttons, checkboxes, dropdown lists or full listboxes. The example use of such enumeration is assigning priority to a document ("high", "normal" and "low") or flagging a case as finished with one checkbox.
- *NoteFile* – replacing the file attachments in the original software, mostly used for digital originals or scanned copies of documents. This field enables to create a link between an object (like document) and a computer file. This link can be multiple or single choice if needed.
- *NoteText* – replacing all text fields from metadata forms, like "subject", "description", "notes" or "document type". The field can be a dictionary field, with all values used in such context available in dropdown list and the ability to add a new value to the dictionary by entering a new text. The field can also be a freeform multiline text field for descriptions or notes.
- *NoteUser* – replacing person designation in the original solution, in fields like "to", "from" and "cc". This field enables to create a link between an object (like document) and a person (which is also an object, so it is possible to create links like "supervisor"). The definition of this field allows for linking multiple users (like multiple addressees) or just one (like one person responsible for this object). It also allows narrowing down the possible people choices to one group (like "employees").

Common features among all *notes* entities are:

- mandatory – absence of the proper note value on the metadata form will make the validation of the form fail. Fields like NoteUser "to" or "from" are usually mandatory on the document creation form;
- immutable – the value can be set only once, usually on the object creation. A good example of its use is the "delivery date" field in case of documents delivered by post and entered to the system on later time;
- object type – the types of objects this *note* can be attached to. Some notes are meant to be attached to documents (like "to", "from" and "cc") while other are describing people ("supervisor") or groups ("due date").

It is possible to further improve the *Notes Framework* by adding new note types. There are also plans to develop a logic API on the *Notes Framework* and allowing for conditional connections between different notes.

Layout configurator

With so many configurable options present, it was necessary to design a flexible user interface solution. In the original software the various metadata forms were carefully implemented in design time and were not prepared to work with dynamic fieldsets like the *Notes Framework*.

Addressing this issue, the layout configurator was deployed as a part of Groups concept. The *Layout* object consists of an XML file listing the fields to display and their positions, and the object type this layout can represent. For each *GroupType* and its accepted object type there is a *Layout* object attached. In the Group view, opening a metadata page for a particular object type launches a form built accordingly to this *GroupType*'s *Layout* for this object type. Each of previously mentioned functional elements – automatic labels, notes from the *Notes Framework* and a few system fields and buttons (like "save" and "cancel") – can be placed on the layout for a valid object type. Each of these elements comes with its own logic, validation schemes and runtime instructions. They work in tandem to provide a functional and highly configurable user interface. The XML files with layout configurations are modified with a user-friendly drag-and-drop editor.

Summary and further considerations

The aim of the major update of the software was to switch from an outdated technology to a modern solution. This provided an opportunity for great structural changes addressing most issues indicated by users and help-desk staff alike. The new concepts and modules replaced the old solution, as the original, simple idea of document classification was expanded during the original software's lifetime and became complicated and unclear. With the necessary update for current operating systems it was possible to get back to that idea, reflect on the changes and design a classification system from scratch, taking into account both users' needs, reflected in modification to established software, and requests from surveys and personal contacts.

With the new, flexible frameworks it is now possible to reconfigure the software according to particular user's needs and requests without tedious redesigning, recompilation and worries about backward compatibility.

Among features yet to be added there are:

- more identifiers to be used in *NamingPattern* entities, especially in connection with the *Notes Framework*;
- mathematical operators in *NamingPattern* counters;
- logic API to the *Notes Framework* providing conditional connections between different notes;
- more note types as the need arise.

The difficult decision of a major technological update was further expanded with multiple architectural changes, but the end product has not only the sum of features made for different users in different instances, but also has the possibility to encompass all other feature requests of the same sort.

Addendum

Original technological specification of the document management system in question:

- MS Access Forms for the user interface;
- MS Access Database for data storage.

Final technological specification of the document management system in question:

- WPF (Windows Presentation Foundation) for the user interface;
- ADO.NET Entity Framework 4.0 backed by MS SQL Server for data storage;
- WCF Web Services (Windows Communication Foundation) and OData (Open Data Protocol) for data delivery.

References

[1] Hou, J.L. and Chan, C.A. A document content extraction model using keyword correlation analysis. International Journal of Electronic Business Management. 2003, Vol. 1, 1.

[2] Azad A.: Implementing electronic document and record management systems. s.l.: Auerbach Publications, 2008.

[3] Rubin M.: Wybrane metody indeksowania i przeszukiwania pełnotekstowego plików w aplikacjach MS Access 97. Techniki Komputerowe. 2/2000.

[4] Kotowski M.: Dokmistrz z Warszawy. PC Kurier. 6/2000.

[5] Przyborowska B.: Problemy tworzenia polskojęzycznych instalatorów aplikacji MS Access. Techniki komputerowe. 2/2001.