A fact approach to automatic application development

Elton Manoku[1], Jan Pieter Zwart[2], Guido Bakema[2]

[1]Freelance IS consultant
[1]emanoku@gmail.com. Tel: +355-69-20-81-566, Tirana, Albania
[1,2]Research and Competence Group Data Architectures & Metadata Management
[1,2]Informatics and Communication Academy, HAN University of Applied Science
[1,2]Beverweerdlaan 3, 6825AE Arnhem, The Netherlands
[1,2]Phone: +31-26-3658271. Fax: +31-26-3658126
[2]janpieter.zwart@han.nl, guido.bakema@han.nl

Abstract. This paper presents an architecture for defining models for application generation from the fact oriented point of view. We explain how to generate application components such as user interface parts, the database and the transactions. This requires a *data use model* that contains pragmatic aspects of the UoD, in addition to the usual *data structure model*, which contains the semantics of the UoD. The single point of definition of these models simplifies their validation by the domain expert and ensures their mutual correctness during the entire development phase. The infrastructure was implemented in a prototype tool, which supports the creation of the application metadata and generates the application components. This prototype tool has an open repository. We are using it for further research in this area as well.

Keywords: Information Modeling, Data Modeling, Application generation, fact oriented approach

1 Introduction

Organizations try to adapt to fast changes in the market to survive the competition. The information systems they rely on therefore frequently have to cope with changing user requirements. A fast application development method is needed to assure users their needs are properly understood and to incorporate these changes into the system.

At the HAN University of Applied Science we have developed a procedure to automate this fast development of information systems. We use a fact oriented conceptual approach, which maintains the participation of the users during the entire development process, so the developer can validate his/her design at any time. The information system model consists of an extended conceptual data model able to include the user requirements for the system.

In this paper, we only briefly sketch the several issues involved in realizing automatic application generation from a fact oriented point of view; each issue would merit a separate paper.

This article is divided into four main sections. Section 1 sketches the context of our work. Section 2 describes how to define a process driven approach from a data perspective, which enables a single point of definition for all metadata for the automated information system in a conceptual layer. Section 3 shows how the automated information system components can be generated from this single point of metadata. Section 4 explains how everything can be implemented in a single repository, based on the FCO-IM repository [1, 2] extended with extra fact types.

2 The extended data model

We distinguish between data related user requirements, described as usual in a *data structure model*, and user requirements that tell us how, when and by whom the data is handled, described in a *data use model*. Both models are integrated in an *extended data model*, stored in a single point of definition, which facilitates the validation of the information system and the automatic generation of the applications.

2.1 Concrete example

To illustrate these matters, we will use a small concrete example. It consists of an information system to support the registration of preferences of students for different study programs that a certain university offers, and the acceptance of the students in one of these programs; see the description in figure 1. This Program Registration example is as small as possible for this paper, but the method has proved to work in larger cases.

An applicant can register for a study program by supplying his/her name, date of birth, nationality, etc, and state up to 3 preferences for a study program.

Lecturers determine the programs on offer, and which kinds of documents are required for each program, such as diplomas or mark lists.

The administration registers the received documents in the system. The administration decides the enrollment deadline for each program as well.

An applicant will be accepted in a study program if it is one of his/her preferences, all necessary documents are present, the enrollment deadline has not yet passed and the number of students accepted is less than the maximum number.

Fig. 1. Starting document describing the UoD of the Program Registration example.

2.2 The data structure model

We use Fully Communication Oriented Information Modeling (FCO-IM) [3] to express the data structure model in, because of the fact approach it takes, and because it is based on a subset of natural language.

The FCO-IM data structure model presents fact types and business rules in a standardized way. The diagram in figure 2 captures all data fact types and basic business rules such as uniqueness constraints, totality constraints, etc. More complex business rules – such as the derivation rules for fact types Acceptance and Current_No_Accepted – are not expressed in the diagram, but supplied in different form as part of the model.

From an FCO-IM data model, we can generate a complete, redundancy free database structure, including the parts that cannot yet be defined with a standard DDL.

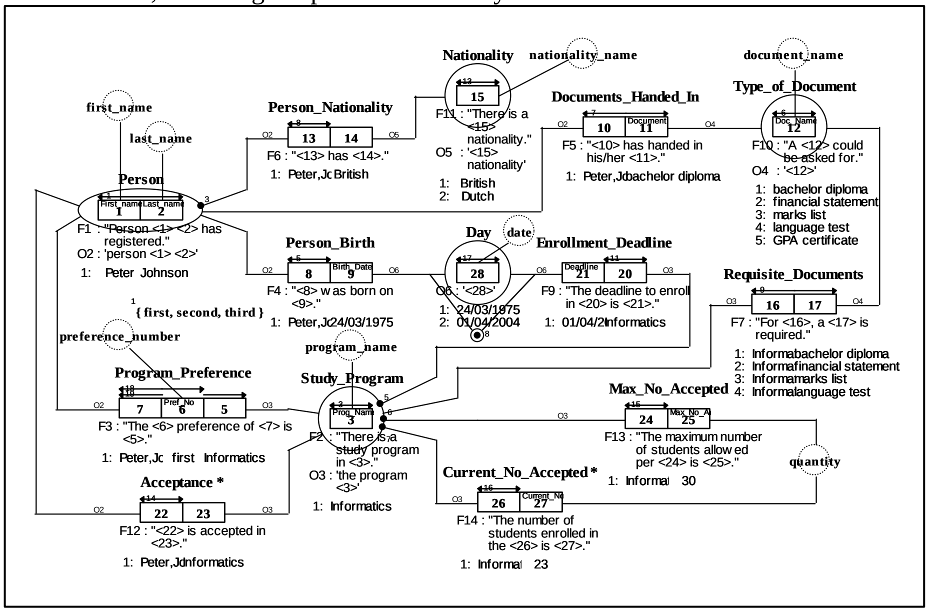

Fig. 2. FCO-IM data structure model for the Program Registration example.

2.3 The data use model

The data use model specifies user related aspects of the information system. It contains user authorization and preference facts, and conditional constraint facts. Examples of these facts will be given below.

It is important to recognize that data *structure* fact types are the primary types of fact to be determined by the developer. The data *use* facts only have meaning in relation to the data structure fact types, and are indeed defined in terms of these data structure fact types. So the data use model can be seen from a completely data oriented point of view.

In the following we describe how to define the main elements of the data use model from a data driven perspective.

Defining user authorization facts in terms of data structure fact types. User authorization facts and preference facts can only be specified fully by referring to data structure fact types. Examples:

A user in the role of applicant has the right to insert facts of type Person.

A user in the role of lecturer has the right to delete facts of type Study_Program.

Examples of user preference facts will be given in section 3.

Defining conditional constraints in terms of data structure fact types. Conditions that influence the processing of data facts can be defined as business rule facts on the data structure fact types and user authorization facts. Examples of such constraints:

An applicant will be accepted in a study program if it is one of his/her preferences, all necessary documents are present, the enrollment deadline has not yet passed and the number of students accepted is less than the maximum number.

A user in the role of applicant has the right to insert facts of type Program-Preference only after he/she has inserted a fact of type Person

All such business rule facts can be expressed as set constraints or negative set constraints [4].

2.3 Merging the data structure model and the data use model

By viewing the data use model from the data perspective, we can integrate both data models, since the elements of the data use model are defined based on the data structure model. We thus create a single point of definition for all metadata, which we call the extended data model. Figure 3 shows this, also illustrating the development process and the operational process of the automated information system.

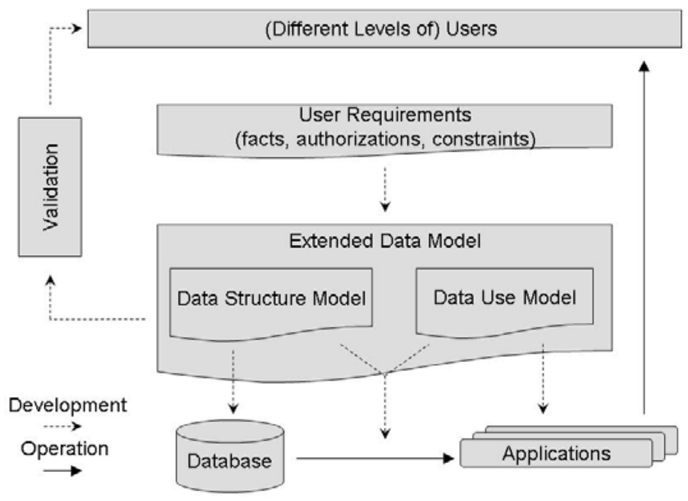

Fig. 3. Extended data model as single point of definition for all metadata.

3 Generating an automated information system

Automatically creating an information system consists of generating the *database structure*, a *set of screens* and a *set of transactions*.

The *database structure* can be generated from the data structure model, via an intermediate Entity-Relationship model or otherwise, as described elsewhere [3, 5]. We describe in somewhat more detail below how to generate the *set of screens*, which allows users to interact in a user friendly way with the data facts. The *set of transactions,* the link between the set of screens and the data facts, is discussed briefly at the end of this section.

3.1 Generating the set of screens

The set of screens supports the users of the system to interact in a consistent way with data facts. The user interface reflects the user point of view on the facts. Creating a set of screens consists of generating *forms* and *user interface objects* within a form. The information used for generating the user interface is present in the extended data model as data fact types, user authorization and preferences facts, and conditional constraints. Below we sketch the steps involved in generating a set of screens.

Clustering elementary fact types. The FCO-IM data structure model (see figure 2) contains elementary fact types, which are to be clustered to form collections of fact types that users find natural. This clustering takes place in two steps. The first step, *grouping fact types*, is determined completely by the data structure alone, whereas the second step, *clustering groups of fact types*, requires user choices.

Grouping fact types. The first step in the clustering process is analogous to the grouping algorithm in FCO-IM [3], but from a broader point of view. Instead of ‘absorbing’ fact types into other fact types [3], they are kept intact but are assigned to a *group of fact types*. Each group contains one fact type as its identifier fact type, and is given the same name as this identifier. The other elementary fact types in the group can be regarded as grouped to this identifier fact type. Each elementary fact type that itself is completely identified by this identifier fact type will be grouped to it. Examples:

1. Each tuple in elementary fact type Birth_Date is identified by a person (uniqueness constraint on role 8), therefore we say that fact type Birth_Date is completely identified by fact type Person, so Birth_Date is grouped to Person.
2. Enrollment_Deadline is grouped to Study_Program.
3. Requisite_Documents cannot be grouped to either Type_Of_Document or Study_Program, since neither identifies Requisite_Documents completely.

Figure 4 shows the result of this grouping process for the Program Registration example. However, of each group of fact types, it shows only the name of the identifying fact type. The group Person actually contains the following 4 fact types: Person, Person_Nationality, Person_Birth and Acceptance. Their presence can be seen from the fact type expressions F1, F4, F6 and F12 below the fact type.

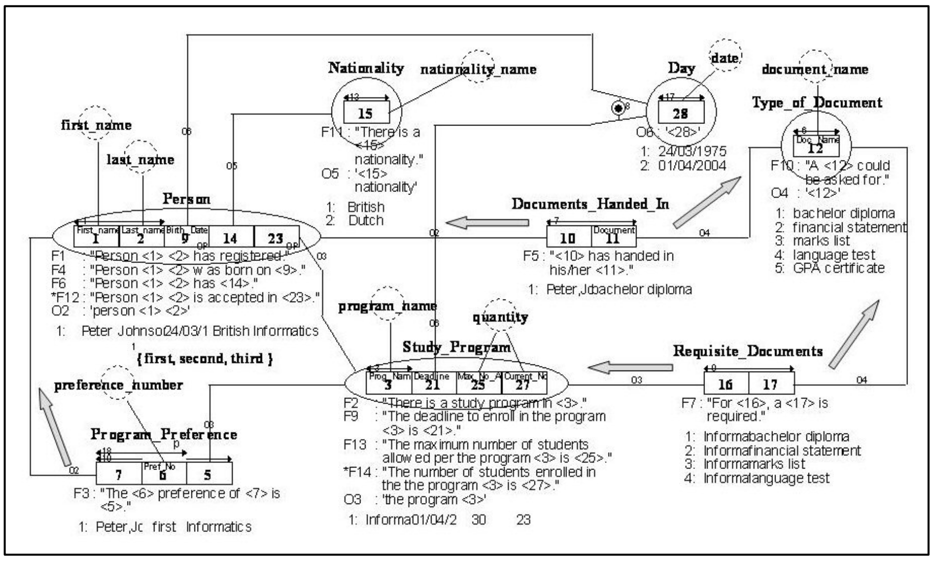
Nationality
nationality_name
15
F11 : "There is a <15> nationality."
O5 : '<15> nationality'
1: British
2: Dutch
Day
date
28
O6 : '<28>'
1: 24/03/1975
2: 01/04/2004
document_name
Type_of_Document
Doc_Name
12
F10 : "A <12> could be asked for."
O4 : '<12>'
1: bachelor diploma
2: financial statement
3: marks list
4: language test
5: GPA certificate
first_name
last_name
Person
First_name
Last_name
Birth_Date
1
2
9
14
23
F1 : "Person <1> <2> has registered."
F4 : "Person <1> <2> was born on <9>."
F6 : "Person <1> <2> has <14>."
*F12 : "Person <1> <2> is accepted in <23>."
O2 : 'person <1> <2>'
1: Peter Johnso24/03/1 British Informatics
Documents_Handed_In
10
Document
11
F5 : "<10> has handed in his/her <11>."
1: Peter,Jobachelor diploma
{first, second, third}
preference_number
program_name
quantity
Study_Program
Prog_Nam
Deadline
Max_No_A
Current_No
3
21
25
27
F2 : "There is a study program in <3>."
F9 : "The deadline to enroll in the program <3> is <21>."
F13 : "The maximum number of students allowed per the program <3> is <25>."
*F14 : "The number of students enrolled in the the program <3> is <27>."
O3 : 'the program <3>'
1: Informa01/04/2 30 23
Requisite_Documents
16
17
F7 : "For <16>, a <17> is required."
1: Informabachelor diploma
2: Informafinancial statement
3: Informamarks list
4: Informalanguage test
Program_Preference
p
Pref_No
7
6
5
F3 : "The <6> preference of <7> is <5>."
1: Peter,Jc first Informatics

Fig. 4. FCO-IM data structure model for the Program Registration example after grouping.

Clustering groups of fact types. The grouping in the previous step could be done without introducing any redundancy. Further clustering is often desired by the users, however. It is possible to cluster groups together if one group partly identifies the other. This can be done in several ways, depending on uniqueness constraints, and so the user can state his/her preferences, which is another opportunity to obtain user validation. Examples (see figure 4):

1. The group Requisite_Documents is partly identified by the group Study_Program and partly by the group Type_Of_Document. It can therefore be clustered together with Study_Program, Type_Of_Document, or both (indicated by the large arrows).
2. Documents_Handed_In can be clustered to Person, Type_Of_Document, or both. The user is free to choose any option (including no clustering at all).
3. Group Preference has two identifiers, one relating partly to Person (uniqueness constraint 2 on roles 7 and 6) and the other to Study_Program (constraint on roles 6 and 5). The user considers uniqueness constraint 2 the *primary identifier* of this fact type (indicated by the little ‘p’ in figure 4). This user choice implies then that it can be clustered to Person only.

Generating forms. Each group of fact types formed in the first clustering step above will be implemented on exactly one form. For each group that is additionally clustered to another group in the second clustering step above, a child-form will be generated, with the form of the group it is clustered to as its parent form.

Forms. The name of the group is also that of the form. One or more user interface objects will be created for each elementary fact type in the group. If the elementary fact type is derivable (such as fact types Acceptance and Current_No_Accepted), then the user interface object(s) will be read-only, otherwise they will support both reading and updating facts.

The identifier fact type of each group will be the default identifier for the form. Sometimes however users would like to have a more ‘natural’ identifier than this. Therefore the identifier can optionally be extended with other fact types that are part of the group. We call such an extended identifier a *human key* [6]. For example: the user wants Person_Birth to be part of the identifier on the form, so a person is identified as ‘Peter Johnson 24/03/1975’, although ‘Peter Johnson’ formally suffices.

Child forms. Choosing which child forms are to be placed on a parent form is really a domain expert choice from the options during the second step of the clustering process. See figure 4: if the user chooses to cluster Person, Program_Preference and Documents_Handed_In, then the parent form Person will contain two child forms Program_Preference and Documents_Handed_In.

For each child form a child-form-preview user interface object is introduced. This object will serve as a gateway for maintaining data about the child form and will offer a preview of child form data.

User forms. The forms explained above are the basis for generating the actual forms for the users. The further form design is based on the intersection of the user authorization rights and the elementary fact types clustered on the form, taking conditional constraints into account as well.

Forms directly accessible from the main menu. Forms that will not be child-forms of other forms will be directly accessible from the main menu. The user can choose to have some child-forms directly accessible as well.

Generation of user interface objects (UIO’s). Every fact type needs a UIO to be accessible in a user friendly way. From a structural point of view, fact types are collection of roles. Some of these roles correspond with the identifier fact type of the group and the other roles are specific for the fact type. A UIO will be created for every specific fact type role. The functionality of the UIO depends on the object type that plays the role. There are two kinds of object types playing roles: lexical object types and non-lexical object types.

Lexical object types (LOT’s). A LOT is the most basic construct in an FCO-IM model, which can be seen as a source of names. Every LOT has an elementary data type and value constraints associated with it, used by the corresponding UIO’s. The UIO’s make sure that only allowed values can be inserted. Examples:

LOT first_name accepts strings of up to 15 characters. The UIO for roles played by *first_name* will be a text box that allows up to 15 characters.

LOT preference_number accepts strings of characters from the set {first, second, third}. The UIO for roles played by *preference_number* will be a list of allowed values from which the user can pick. It can be a combo box or a list box.

Non-lexical object types (NLOT's). An NLOT can be seen as a compound data type defined in the model. The UIO's that represent the NLOT's are derivations of a generic UIO that we call the lookup object. It is possible to define a specific UIO for every NLOT. The lookup object encompasses the functionality that is common for all NLOT's. It displays the list of the fact types involved in the human key and allows the user to pick one of the values. Example: There is an NLOT Study_Program identified by Program_name. The user likes to have the deadline date in addition to the program name as human key. In the lookup object for Study_Program there is also a list of all programs registered.

User preferences. As shown above, the user interface is mainly generated from the information in the data structure model, the authorization rights and the conditional constraints. However there are also facts about the user interface that depend only on user preferences. These preferences are confined within the boundaries defined by the structure, authorization rights and constraints. These user preferences facts are seen as part of the model and must be stored as well in the single point of metadata definition. Below we mention some of the most important types of user preferences.

User preferences that have to do with UIO colors, the way they are organized in a form, the order of access of elementary data fact types, etc. These preferences can be determined by interviewing the users, but we always supply a default choice. Example: The order of access to UIO's in a form can be derived from the order of the elementary fact types in their collections.

User preferences that have to do with UIO's that have more than one alternative of display. Example: LOT preference_number has more than one representation for its UIO: drop down list, option buttons, or list.

User preferences that have to do with the human key definition. Only the domain expert can decide how to extend the minimal key with other fact types.

User preferences that have to do with lookup child-form object definition. The domain expert can decide which fact types to include in a preview of a child form.

3.2 Generating the transactions

Transaction processing assures that the correct transaction is executed in the correct user form by considering all constraints and authorization rights. Facts are entered into the system through the UIO and stored in the database, which is a relational model. A transaction does the mapping between the UIO in the user form, the database structure and the data fact types. For every user form generated by the basic form, transactions that insert, delete and update facts are generated as well. A transaction associated with a form considers only the collection of facts types clustered together and takes the user authorization rights and conditional constraints into account. A transaction involves:

- checking constraints involving new candidate facts and all existing facts
- executing one or more main insert/delete/update statements for storing new facts
- executing other relevant insert, delete or update statements
- checking complex constraints that cannot be checked by the DBMS. This involves conditional constraints as well.

From the above description it is clear that the complete extended data model (both data structure and data use model) is needed to automatically generate an information system.

4. The implementation

The implementation of these ideas consists of two main tracks:

1. Automation of the development process

This involves the set of tools used to translate the user requirements into metadata stored in the repository. These tools must also support the validation process. The tools involve: the data Modeling Tool to create the data structure model (for which we use CaseTalk [2]), and the Application Bridge that supports the developer for interacting with the information grammar to extend the information grammar with application definition related facts.

2. Automation of the generation process

This is the Application Engine. It understands the extended data model repository (the extended FCO-IM repository), generates the user interface, controls the user rights and does the transaction processing. See figure 5.

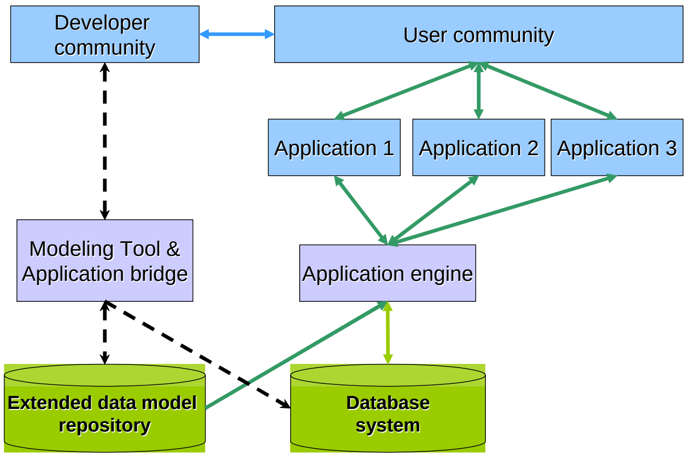

Fig. 5. The development and generation processes.

Conclusion

This paper touches the main elements of application generation from a fact oriented point of view. Advantages of taking this approach versus the more classical way of considering application definition as an extension of the relational model are:

- More consistency in the definition of concepts. In the fact approach everything is considered as a fact.
- User authorization and other constraints are defined upon fact types and/or facts. This reduces mistakes made during the determination of the constraints and user authorization and makes their validation easier.
- It introduces the definition of user authorization and conditional constraints from the start in the development process.

All elements in the application generation process mentioned in this paper are currently being investigated in greater detail.

References

1 Bakema, Guido: FCO-IM Repository, Course materials, HAN University of Applied Science, Arnhem, the Netherlands (1995).

2 BCP Software: CaseTalk, FCO-IM modeling tool, see http://www.CaseTalk.com.

3 Bakema, Guido; Zwart, Jan Pieter; Lek, Harm van der: Fully Communication Oriented Information Modeling (FCO-IM), 2002. The book can be downloaded for free from http://www.casetalk.com/php/index.php?FCO-IM%20English%20Book.

4 Bakema, Guido: Negative SQL Statements, private communication (2002).

5 Manoku, Elton; Bakema, Guido: Integrated Tool support for Datawarehouse Design, Journal of Conceptual Modeling, issue 34, January 2005.

6 Luursema, Eddy: Human Key, private communication (2004).