A new approach to adaptive data models

Ion LUNGU, Andrei MIHALACHE
Faculty of Cybernetics, Statistics and Economic Informatics,
Academy of Economic Studies, Bucharest, Romania
ion.lungu@ie.ase.ro , andrei@mdata.ro

Over the last decade, there has been a substantial increase in the volume and complexity of data we collect, store and process. We are now aware of the increasing demand for real time data processing in every continuous business process that evolves within the organization. We witness a shift from a traditional static data approach to a more adaptive model approach. This article aims to extend understanding in the field of data models used in information systems by examining how an adaptive data model approach for managing business processes can help organizations accommodate on the fly and build dynamic capabilities to react in a dynamic environment.
Keywords: *adaptive data model, dynamic capabilities, data models*

1 Introduction

Data dominates every information system and if data structures are properly chosen and organized things go well, any algorithm is almost always understood by itself to be optimally built to perform. Algorithms are essential in programming, but data models will always fill the central place [1].

The business environment is the sum of all those factors which are available outside the business and over which the business has no control. Some of these factors can include objects such as: clients, suppliers, competitive companies, investors and owners, improvements in technology, laws and government activities, market, social and economic trends.

2 Data models

A *model* can be defined as a simplified abstraction of a complex reality, highlighting the essentials and ignoring the details.

Data modeling is a method used to define and analyze data requirements needed in business processes deployed in companies. Software applications store data in order to use them in the future. When data is saved, most of the times a relational database is chosen due to performance and accessibility (data is understandable). The *data* term refers to facts that characterize objects or events that can be recorded and stored in a computer system and it has significance and meaning for users.

Data governance is a set of processes that ensures that important data assets are formally managed throughout the enterprise and take into account data definition and data integrity constraints in the data model [2].

3 Data model types

The 1975 ANSI/SPARC data architecture study group divided database-centric systems into three models:

- *Internal model*: describes the logical data structures and may contain logical descriptors of the collections, attributes, XML markers, etc. These models are represented in accordance with the requirements of a particular technology implementation using flowcharts.
- *Conceptual model:* represents the scope and semantics of the classes designating entities of interest to the study area and assertions about associations between these entities. They are represented by the conceptual diagram.
- *External model*: defines how data is stored. These models contain

partitions, tablespaces, indexes etc. and it is represented through the physical schema.

Looking back to the late 1960s, IBM launched the *hierarchical model*, together with data manipulation language DL/1. Hierarchical database systems organize data as a collection of trees. All recordings have an owner or root (one and only one), and thus all other records have a single parent.

To locate a certain record, you must travel the path from root parent of the tree to the level where the desired child is located. Access to data in hierarchical databases is achieved by low-level calls that programmers write to sail records from the root towards the leaves of interest. Therefore, the programmer must know the physical representation of the database.

This data model can be used in systems containing data that can be organized hierarchically, without compromising the information (e.g.: **Fig. 1**).

Hierarchical databases support two means of representing information: specific records containing data and records containing the type of parent-child links that defines the relationship 1:N between one parent and N child records.

This approach has major limitations due to restrictions of data representation. The data structures which are not represented hierarchy by default is difficult, if not artificially structured in such database.

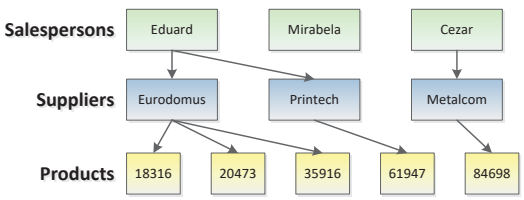

Fig. 1. Example of hierarchical data model

The *network model* was originally created by Bachman for General Electric, where he developed the first commercial database system (IDS - Integrated Data Store) in 1964.

Data modeling in a network database is different from the equivalent hierarchical approach. Networked databases arrange data in a directed graph and use a standard navigation language.

The model has brought new opportunity to move the access from a specific point of a set of data directly to another record in another data set.

Network databases provide an effective pathway to access data and are capable of representing any data structure containing simple types (such as integers, real characters and strings). This is achieved by using different types of mapping mechanisms known as *sets*.

A *set* is a container of pointers identifying the type of data set that can be accessed from the current record. These sets are defined by standard CODASYL: sets of system (single), multimembers or recursive. Using these sets, database designers and programmers can represent and navigate relationships 1:1, 1:N and N:M.

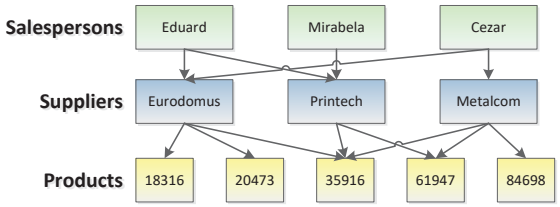

Fig. 2. Example of network data model

To access the data, the programmer must know the physical representation of data and use a low-level navigation language.

This approach to system database is more flexible than the hierarchical model, but the programmer must know the physical representation of data you can access them. Therefore applications using a network database need to be altered with every change in the database structure.

The *relational model* provides a different approach to data storage and it was first represented by Edgar F. Codd in 1970 [3]. In a relational database, all data are represented as simple tabular data structure

(relationships) that are accessed using a high-level non-procedural language.
This language is used to achieve the desired relationships and datasets. Thus, the physical implementation of the database is hidden, and the programmer does not need to know the physical implementation to access the data. In 1974 the name was proposed SEQUEL for high-level non-procedural language, which was later changed to SQL. In 1986 ANSI committee X3H2 accepted as standard ANSI SQL language.

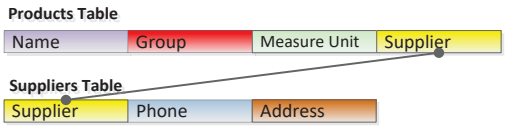

Fig. 3. Example on an instance of relational data model

Relational databases connect different data files by using key fields or some common data (e.g.: **Fig. 3**). Records are stored in different tables or files that are composed of rows and columns. In the databases terminology, tables are called "relations", the rows are called "tuples" and the columns are called "attributes".
The advantage of a *Relational Database Management System* (*RDBMS*) is that users and programmers do not have to know the data structures or pointers. Tables and rows are much more comprehensible than pointers and pointers that point to records.
The downside is that some orders retrieval requires more processing time compared to other database models.
SQL (*Structured Query Language*) is a language based on the declarative transformations as opposed to specific languages based on navigation.
The relational approach separates the physical implementation of the program database. The programs are less sensitive to changes in the physical representation data, linking data and metadata in the database. Application development is more efficient and independent of changes in the physical representation. This is the reason why SQL and relational database systems are widely used: due to the separation of physical and logical representation.
The relational model is sustained by the *entity-association model* whose purpose was not to be implemented, but to represent the data at the abstract and conceptual levels used in computer systems. It was defined in 1976 by P.S. Chen and its specification does not impose any specific data patterning or processing [4].
The *entity-association model* provides an overview and classification of terms used and their relationships, holistic for an entire system or just one area of interest.
The *Enhanced Entity-Relationship Model* includes extensions of the model of Chen and allows the definition of subtypes of a type of entities that inherit attributes from the type of entity that you extend (which in this context is called super type) and additionally attributes their significance. In terms of this model, an entity is an object that exists and can be distinguished from other similar objects [5].
In terms of standard construction of databases, an entity may correspond to a record and its attributes correspond to fields registration. This model is implemented physically, but is used in the analysis of information systems at the logical level.
The *object-oriented model* allows "objects" depositing as elements in the database. An object is composed of text, sound, images and actions that can be applied to the data. Hierarchical, network or relational data models, storage allow only numeric data and text, while object-oriented data model may additionally contain multimedia data like images or video.
The *object-oriented database management systems* provide persistent objects, including associations between objects, and methods. A basic concept of object-oriented model is defined orthogonal persistence [6] by three principles:
The principle of independent persistence: the lifespan of a program is independent of

the data they manipulate. Programs that manipulate data in the short and long term look the same.

The principle of data types orthogonality: All data objects must have full persistence, regardless their type. There are no special cases for items not to be allowed to have a long lifespan or not to be transient.

The principle of persistent identification: Choosing how to identify and supply the items is orthogonally defined to the persistent universe in the system. The identification mechanism for persistent objects is independent on the type of system.

Objects' persistence, including orthogonal persistence, is often achieved through the concept of persistence, when accessibility makes a persistent object if it can be accessed from a persistent root.

This is a completely different approaches grid / tree using language low navigation, and approaches relational or object-relational using a high level language (*HLL - High Level Language*) for navigation, query and manipulation or combined with some SQL data definition language (*DDL - data definition language*).

The *object-relational model* extends the database systems relational model to add concepts from the object-oriented approach and get more complex object structures and rules and yet remain open to other systems. An Object-Relational Database Management System - ORDBMS is most simply defined by the equation in **Table 1**:

Table 1. Defining object-relational model based on object-oriented and relational models

ORDBMS = ODBMS + RDBMS = (O + R) × DB × MS

At the logical level, an ORDBMS is a management system MS that applies methods to process data structures from the database DB and complies both an object O and relational R concepts.

An *object* is an entity with a clear role in the system, characterized by state, behavior and identity. Upon a certain object we can take action that on her own turn can trigger or perform another action. The object can be concrete: a tangible and visible entity, for example a person, place, thing; an abstract entity with it as a concept, an event, a department, marriage, idea; or an artifact of the design process, for example: user interface, control, planning [4].

Any object exposes its behavior through operations that may affect his or another object's state. The state of an object is defined by the values held by properties at a time. The behavior shows how an object acts and reacts to events.

An operation is a simple action performed by an object on another object to get an answer. Operations performed by an object or performed on an object, implemented in a programming language are called methods.

Classes which have links with specific restrictions operations and object-oriented approach, define the object-oriented data model.

The need of persistent data has evolved from sequential files to structured files, network databases, hierarchical databases, RDBMS, and recently into ORDBMS and OODBMS offering more controlled and flexible storage, interface, and transactional capabilities on complex objects and structures.

Fig 4 represents side by side all resembling and complementary concepts and characteristics in data models evolution at conceptual, logical and physical levels.

Over time, there were developed several conceptual models specific to each databases management systems, each with different capabilities, both in terms of organization, data modeling, and access.

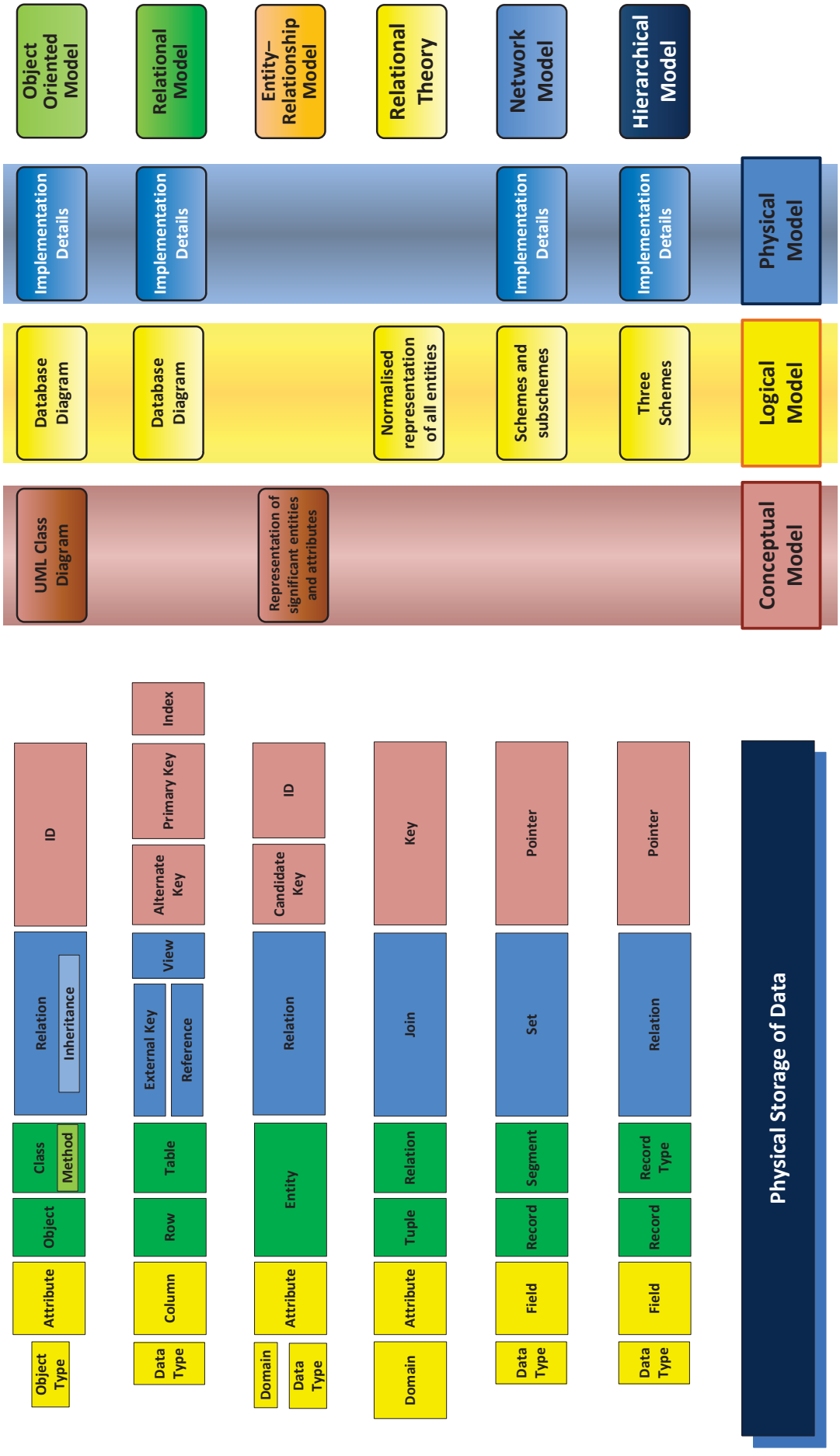

Fig. 4. Correspondence of the data models in database systems evolution

Essentially, data models should allow different applications to be able to share the same data. However, frequently achievement and maintenance of information systems cost more than would be required, and data models, as a result of weak implementation, become an obstacle for business processes rather than act as a support mechanism.

4 A new adaptable data model

The main role of data models is to ensure data compatibility, a necessary and sufficient framework that allows different applications to share the same structures, to store and access data [7].

This paper aims to define and implement an adaptable data model to assist the process models that can be modified on the fly without recompiling modules application. Thereby, at runtime the system allows adding new types, changing existing relations between defined data types and methods, functions and procedures for system processing.

To achieve this, the application must have the ability to work with metaobjects, to instantiate containers of objects that assist workflows [8].

Most applications are too inflexible to keep pace with business processes they support. Built on an architecture with three levels (e.g.: **Fig. 5.**), these applications have two major problems: "components sharing" and "application integration".

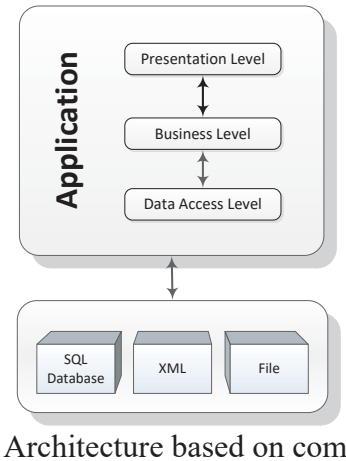

Fig. 5. Architecture based on components

Sharing components within heterogeneous platforms is difficult, if not impossible. When component-based applications need to exchange data, most of the time a designated user must take data manually from one application and put them in the other one.

Adaptability is a characteristic of a system or process. In organizational management, adaptability is generally recognized as the ability to change themselves or other objects to match the changes occurring [9].

Service Oriented Architecture – SOA is an abstract framework that turns business applications into individual business functions and processes. This collection of services is built using compliant standards for systems design integration in real time.

The basic principles of SOA: reuse, granularity, modularity, composability, and interoperability, enable creation of interconnected services between information systems that use middleware, to exchange SOAP (Simple Object Access Protocol) messages.

The *SOAP* standard represents the starting point for messages exchange and exposes behavior of objects, using web services.

SOA makes it possible to loosely couple and reuse functions from processes between different types of platforms [10].

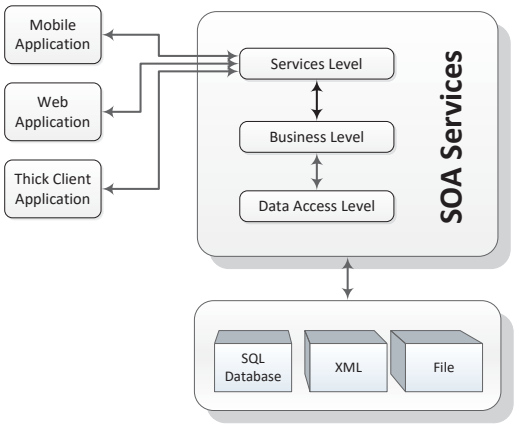

Fig. 6. Service Oriented Architecture

Figure **Fig. 6** exhibits an opposed vision to component-based applications. The Service Oriented Architecture exposes the third level using services. This level describes its interface using SOAP and WSDL (Web

Service Definition Language) and exposes a universal interface which can be used by any interface for users (presentation layer) regardless of device or platform.

Data Oriented Architecture – DOA is a result of "loosely coupled" software components with data-oriented interfaces that enable systems' integration using standards-based communication middle-ware infrastructure [1].

A high level example of WSDL of a supply process that includes the order data model and methods for orders as port types is listed below:

```
<?xml version="1.0" ?>
<definitions name="CerereOferta"
targetNamespace="http://example.com/a
     gentvanzari/wsdl"
xmlns="http://schemas.xmlsoap.org/wsd
     l/"
xmlns:bpws="http://schemas.xmlsoap.or
     g/ws/2003/03/business-process/"
xmlns:plnk="http://schemas.xmlsoap.or
     g/ws/2003/05/partner-link/">

<!-- data types and messages -->
<definitions xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/
envelope/"
xmlns:xsd="http://www.w3.org/2001/XML
Schema"
xmlns:xsi="http://www.w3.org/2001/XML
Schema-instance"
xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/
encoding/"
xmlns:tns="urn:docs_wsdl"
xmlns:soap="http://schemas.xmlsoap.or
g/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.or
g/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsd
l/" targetNamespace="urn:docs_wsdl">
<types>
   <xsd:schema
targetNamespace="urn:docs_wsdl">
      <xsd:import
namespace="http://schemas.xmlsoap.org
/soap/encoding/"/>
     <xsd:import
namespace="http://schemas.xmlsoap.org
/wsdl/"/>
     <xsd:complexType name="order">
       <xsd:all>
         <xsd:element name="id"
type="xsd:int"/>
         <xsd:element name="number"
type="xsd:string"/>
         <xsd:element name="dueDays"
type="xsd:int"/>
         <xsd:element name="issueDate"
type="xsd:date"/>
         <xsd:element name="delivery"
type="xsd:date"/>
        </xsd:all>
      </xsd:complexType>
      <xsd:complexType name="orders">
         <xsd:complexContent>
            <xsd:restriction base="SOAP-
ENC:Array">
               <xsd:attribute ref="SOAP-
ENC:arrayType"
wsdl:arrayType="tns:order[]"/>
            </xsd:restriction>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:schema>
</types>

<!-- port type definitions -->
<portType name="InterfataAgentVanzari">
   <operation name="CerereOferta">
      <input message="cerereOferta"/>
      <output
     message="confirmareOferta"/>
   </operation>

   <operation name="AnuleazaComanda">
      <input message="cerereAnulare"/>
   </operation>

   <operation name="LivreazaComanda">
      <input message="cerereLivrare"/>
      <output
     message="confimareLivrare"/>
   </operation>
</portType>

<portType name="InterfataMerceolog">
   <operation name="NotificareAnulare">
      <input message="mesajAnulare"/>
   </operation>

   <operation
     name="NotificareExpirareTimp">
      <input
     message="mesajExpirareTimp"/>
   </operation>

   <operation name="NotificareReceptie">
      <input
     message="receptieMateriale"/>
   </operation>
</portType>

<!-- partner link types definitions -->
<plnk:partnerLinkType
     name="SerciciuAgentVanzari">
   <plnk:role name="AgentVanzari">
      <plnk:portType
     name="InterfataAgentVanzari"/>
   </plnk:role>
   <plnk:role name="Merceolog">
```

```
      <plnk:portType
     name="InterfataMerceolog"/>
   </plnk:role>
</plnk:partnerLinkType>

<!-- properties definition -->
<bpws:property name="IDrezervare"
     type="xsd:string"/>

<!-- aliases for properties are
     omitted -->
</definitions>
```

Keeping the metadefinition of data and methods inside the application provides an adaptable data model to assist the process models that can be modified on the fly without recompiling modules application. All messages delivered using SOAP over HTTP are self-described and clients can interpret all data and methods using the contained definition.

Low coupling abstract interfaces offered by WSDL (including data types definitions, messages and port types) represent the premise and advantage for building a flexible system with a design meant to change on the fly.

5 Conclusions

Companies are performing in a dynamic economic environment and information systems assisting their business processes are limited by high degree of coupling in three major areas: systems are limited by coupling between modules and their interaction with other systems; data model is fixed at compilation time and does not allow deviations or subsequent adjustments; and workflow models do not allow improving and evolution of the business processes.

Dynamic exposure of all evolving process models and adaptive data models by WSDL interfaces creates the premise of a versatile system in which each object, rule, function, or service interface can be changed in a shorter time. The aim is to optimize the process to obtain a value or quality.

Therefore, the essence of SOA is to dynamically link resources with chain transformations, while the ground of DOA is to expose the data and hide the code.

Acknowledgment

We owe thanks to all the members of the Department of Informatics, for the sessions and Communications Conference on Informatics where we could present case studies and also published the results of our research. It's always a pleasure working with members of the Database Collective Faculty of Cybernetics, Statistics and Informatics.

References

[1] I. Lungu, A. Mihalache – *Enterprise modeling and software engineering for information systems improvement*. Studii şi Cercetări de Calcul Economic şi Cibernetică Economică, pp. 18, Nr 2/2010, ISSN: 0585 – 7511, EISSN: 1843 – 0112

[2] Data Governance Institute, *Definitions of Data Governance*, http://www.datagovernance.com/adg_data_governance_definition/.

[3] E. F. CODD, *A Relational Model of Data for Large Shared Data Banks*, IBM Research Laboratory, San Jose, California, 1970, https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[4] I. Lungu, A. Bâră, C. Bodea, I. Botha, V. Diaconiţa, A. Florea, A. Velicanu, *Tratat de baze de date – Baze de date. Organizare. Proiectare. Implementare*, 2011, ISBN 978-606-505-481-3;

[5]I. Lungu, M. Velicanu, *Database Systems – Present and Future*, Informatica Economica Journal, Inforec, ISSN 1453-1305, EISSN 1842-8088;

[6]M P. Atkinson and R. Morrison, "Orthogonally Persistent Object Systems", *VLDB Journal* 4 (3), 319-401, 1995, ISSN: 1066-8888

[7]I. Lungu, Ghe. SABĂU, M. Velicanu, M. Munten, S. Ionescu, E. Posdarie, D. Sandu, *Sisteme Informatice. Analiză, proiectare şi implementare*, 2003, ISBN 973-590-830-1;

[8]I Lungu, A. Mihalache, "Adaptable

Enterprise Modeling – A New Challenge for Collaborative Data and Process-Aware Management Systems". *Proceedings of the World Multiconference on Mathematics and Computers in Business and Economics* (MCBE '10), Iaşi, România, pp. 261–266, WSEAS Press 2010, ISSN: 1790-2769

[9] K. Andersen, N. Gronau, *An Approach to Increase Adaptability in ERP Systems*, In: Managing Modern Organizations with Information Technology: Proceedings of the 2005 Information Recources Management Association International Conference, 2005, ISBN: 978-1-59140-822-2;

[10] E. A. Marks, *Service-Oriented Architecture Governance for the Service Driven Enterprise*, John Wiley & Sons, 2008, ISBN 978-0-470-17125-7;

[11] A. Mihalache, C. Vintilă, A. Cornescu – *A New Model Approach for Business Applications in Enterprise 2.0 Environment*, In Proceedings of the World Multiconference on APPLIED ECONOMICS, BUSINESS AND DEVELOPMENT (AEBD '09), June 2009, Tenerife, Canary Islands, Spain, pp. 192–195, WSEAS Press 2009, ISSN: 1790-5109.

Ion LUNGU graduated from the Faculty of Cybernetics, Statistics and Economic Informatics of the Academy of Economic Studies in 1974. He got the title of doctor in economy in the specialty economic informatics in 1983. He has been directing graduates who study towards getting a doctor's degree since 1999. At present he is a professor in the department of the faculty of Cybernetics, Statistics and Economic Informatics of the Academy of Economic Studies of Bucharest. He had documentary activity and specialization with the Eindhoven Technical University of Holland, the Economic University of Athens and Economic University of Milan. His domains of work are: informatics systems and databases. Among his books are: "Databases, organization, design and implementation", (1995), "Information Systems for Management" (1994), "SGBD Oracle Applications" (1998); "Let's learn Oracle in 28 lessons" (2003), "Database systems" (2003), "Information Systems – Analysis, Design and Implementation" (2003).

Andrei MIHALACHE has a background in computer science and is interested in database related issues and enterprise modeling techniques. Currently, he is a PhD candidate in the field of Economic Informatics at the Academy of Economic Studies. His research interests include: enterprise data model, adaptable information systems, business process improvement, and web collaborative technologies.