Вісник Харківського національного університету
Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»
№ 847, 2009, с.87-97

УДК 004:378.1

Integrated Environment for Software Development and Analysis

L. Globa, T. Kot, D. Lysenko
National Technical University of Ukraine, Information Telecommunication Networks Department, Kiev, Ukraine

The article deals with the compare of standard, new suggested information system (IS) development logical stages and realizing IS like system based on multithread processing. Developed systems, functioning in the telecommunication environment, which provides access to different services working separately and often being physically distributed, can't be designed as single thread application. Standard and suggested stages are integrated; new CASE-toolkit is being developed for high-quality IS development in the shortest time. The main features of such toolkit is support of prototyping, early testing, analysis, easy reengineering, computing processes in distributed IS optimization and multithread computing in global environment including wireless channels.

Introduction

Distributed information systems (IS) successful design while their development defines the following development stages. A lot of projects, dealing with distributed systems development and implementation fail because of the mistakes, made on design stage.

The scientific work purpose

- Technologies and methods research, analysis and improvement, which are the basis of CASE-tools;
- Efficiency increase of informational and computational resources application in telecommunication environment by multithreading and distributed computing application;
- Information system (IS) development and reengineering process reorganization, which provides total efficiency increase of the information system development and implementation;
- Prototyping, early testing, analysis, computing processes in distributed IS optimization methods development and their realization in software toolkit.

While complex IS development, the business-process description and management, workflow support and software components integration became the most important and topical tasks to be solved [1]. For these purpose quite widespread and convenient CASE-tools are used.

Nowadays it's impossible to imagine modern IS development, implementation and maintenance realized by one specific software tool. Great variety of functions implementation for the system functioning requires a few software tools integration into unified software toolkit. The suggested approach is oriented on network

distributed applications development. Advanced and complex IS realization make the demand for new generation development and support toolkit.

The paper includes suggestions to integrated informational and computational recourses toolkit concept [2] extension for its application in multithread systems [3]. The toolkit gives the possibility to design information system, efficiently working in heterogeneous environment.

For IS general effectiveness increase parallel business processes execution is implemented to IS functioning. It is realised by IS design as multithread one.

Toolkit includes designing tool FFD Designer [4], uses graphical components, allowing system interface, functionality and parallel business process design. Mathematical base of parallel business processes design, analysis and optimization is graph model and finite state machine (FSM) theory mathematical apparatus.

The graph model is used for computational recourses running. It is based on a dynamic forming the effective graph model. It is used while computing threads, running in global heterogeneous environment.

There are some wellknown CASE-tools for distributed information system (IS) designing. But the parallel executing processes while networking are not taken into account by any of them. For realizing these processes some approaches were suggested in this article.

The distributed information system works in telecommunication network, where servers of computational and informational resources function in coordination [5]. Server of information resources includes databases, html, xml- pages and others. Server of computational resources consists of applications and computing threads.

The methodology of distributed IS development bases on the main approaches allowing step-by-step deployment of complex information systems. These approaches are called S2I2 principles:

- simultaneity of system development, implementation and using under conditions of contributive simultaneous work of all project participants: both developers and users;
- standardization of all typical components of the IS informational processes;
- integration as method of the separate components, subsystems and systems organization into the system, which provides their coordinated and purposeful interaction, and also supporting the system business processes execution;
- intellectualization as method for computing of necessary analytical and statistical data.

There are some modified stages of software development. The suggested approach to software development lifecycle is shown on fig.1.

Distributed IS development requires 3 basic components working in global environment:

- informational resources;
- computational resources;
- business-process scenario database.

The suggested toolkit includes tool "FFD Designer", used for IS design and toolkit "core", used for IS business processes execution.

Toolkit "Core" development can be realized by following tasks fulfillment:

1. System design (business processes description) using forms and functions, forms and functions diagram (FFD)
2. FFD translation to FSM
3. FSM functioning realization
4. Work with FSM
   - Analysis
   - Testing
   - Optimization
5. Optimized FFD development, using optimized FSM translation to FFD

Forms and functions description

It is suggested to divide designed system in two levels: representation (interfaces) level and functionality level, described by introduced objects [5]: forms and functions. Such approach is the basis of early prototyping, testing and reengineering realization.

For analysis, testing, and optimization realization while IS development, the mathematical apparatus of finite state machine (FSM) theory is suggested to apply. It allows:

1. Having inputted any impact sequence to designed system enter, to get system state after impact, for its analysis
2. To optimize designed system by finding bottleneck in system prototype and to develop optimized prototype.

Form and Function Mathematical Models

There are two types of the functions: the executive function and the function-prototype (dummy function).

The executive function - this is usual function. There is the **executive function** mathematical model (1) and it structural model is shown on fig.2-b.

$$f = (E_{form}, E_{func}, E_p, x_1, ..., x_n, y_1, ... y_m) \quad \text{, where} \qquad (1)$$

E_{form} is an activity point of form, the function is called by it. It uniquely defines the form

E_{func} is the point of function activity, it belongs to the function and transfers data to the another object (form or function)

E_p is the activity point of the external object (form of function), the control is transferred to it after the current

function execution

$x_1,...,x_n$ are the input function parameters and $x_i \in X$, - the parameters admitted regions

$y_1,...y_m$ are output function parameters, and $y_i \in Y$, - the parameters admitted regions.

Tool	The software development stages	Results
*Special data base**	Specifications definition	*Specifications data base*
*Special CASE-toolkit**	*Early analyses and design*	*IS prototype* • *IS interfaces* • *IS functions*
*Special CASE-toolkit**	*Early testing*	*Correct IS prototype* • *IS interfaces* • *IS functions*
*Special CASE-toolkit**	*Optimization*	*Optimized IS prototype* *Parallel business processes scenarios in database*
*Special CASE-tools for final designing**	*Optimized IS design*	*Final IS prototype and* • *IS interfaces* • *IS functions* *Parallel business processes models in database*
Rational Rose, Together, other CASE–tools	Coding	1. IS wireframe code 2. IS complete code
Tools for testing	Testing	IS correct code
*Special CASE-toolkit and data base for re-engineering**	Deployment and maintenance	Functioning IS

**Italic style – suggested approach*

Fig. 1. The suggested approach to software development lifecycle

The function-prototype (dummy function) is for interface testing (it has the same function mathymatical model but it has no functionality). There is the function-prototype mathematical model (2).

$$f^{prot} = (E_{form}, E_{func}, E_p, x_1,...,x_n, y_1,...y_m) \quad (2)$$

Function-prototypes are needed for whole IS project integrity verification. They are being replaced by the real compiled functions, while IS step-by-step implementation. Function-prototypes must satisfy the following requirements:

- to have the same activity points as executive function does: E_{form} , E_{func} , E_{p} ;
- to control input parameters correspondence to tolerance range;

There is the form mathematical model (3) and it structural model is shown on fig.2-b.

$$F = (N, E_1(x_{11},...,x_{1N}),...,E_N(x_{N1},...,x_{NM})) \quad \text{,where} \tag{3}$$

N	-	number of activity points;
$N_1,...,N_N$	-	the activity points;
$x_{11},...,x_{1N}$	-	the function input parameters E_1 ,
$x_{iN} \in X$	-	parameters admitted regions.

All types of functions can be joined into the new structure called module (fig. 2 c).

Suggested graphic elements business-process design

For IS running in parallel mode some modifications were done in traditional business-process model. Special graphic elements were added to the modified business-process model, it's shown on fig.3.

While business-processes description, the next parameters are considered:

- marks for the form formatting;
- input and output data format of form;
- "specified execution time".

The ability if working with table, thread and other specified data (SQL requests, date, time) is also considered.

The business-process diagram (fig.3) use the elements (fig.2) for business-process running description in multithread mode.

The method of toolkit "core" realization and its functioning is discussed by using the example diagram (pict.3).

The presented diagram of designed system includes four forms. User can sequentially pass on forms F1 F2 F3 and also stop passing and return to initial system state (F1) using F4, which is used for navigation.

In the discussed case, every transition on form entail corresponding function call. Transition is realized as soon as form is handled or function is executed.

Transition between entry and exit points is presented by arrows. Every transition is brought into accord with computing thread identifier (ID). Two special transition types are represented on the diagram: computing thread beginning, represented with bold arrow, computing thread end , represented with square. Two computing threads are used in system example, , represented as 1 and 2, and every transition is brought into accord with one of these threads.

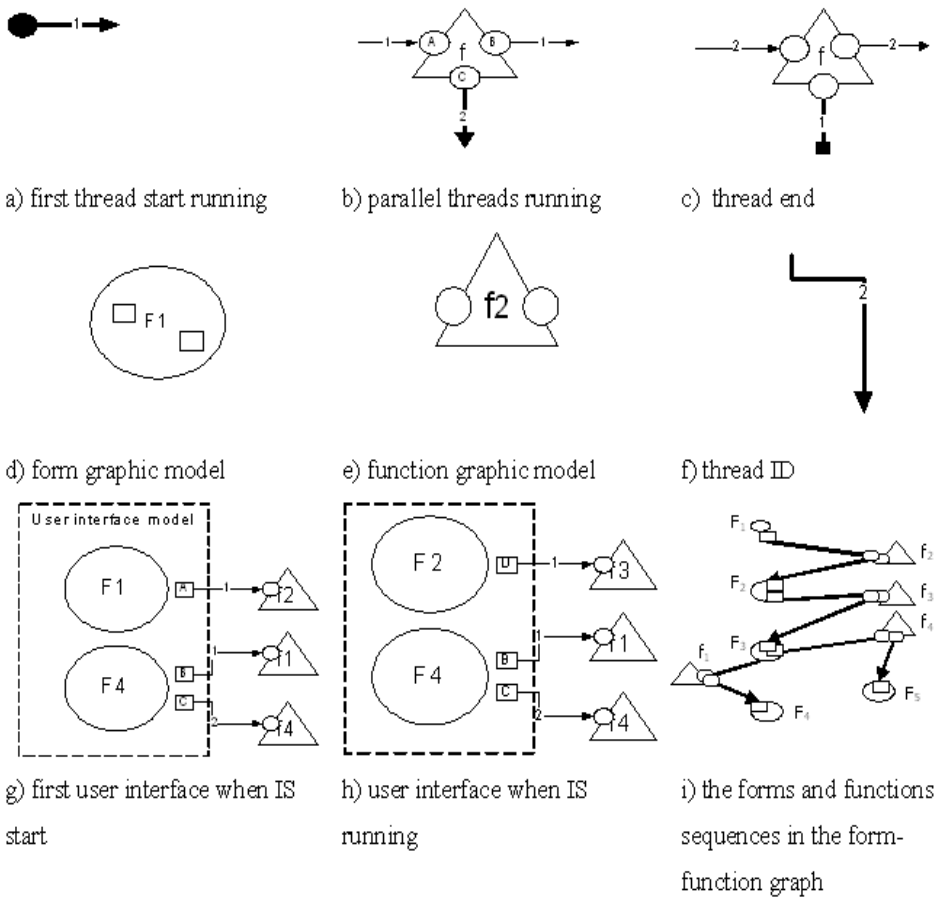

Fig. 2. Additional graphic elements for modified business-process design

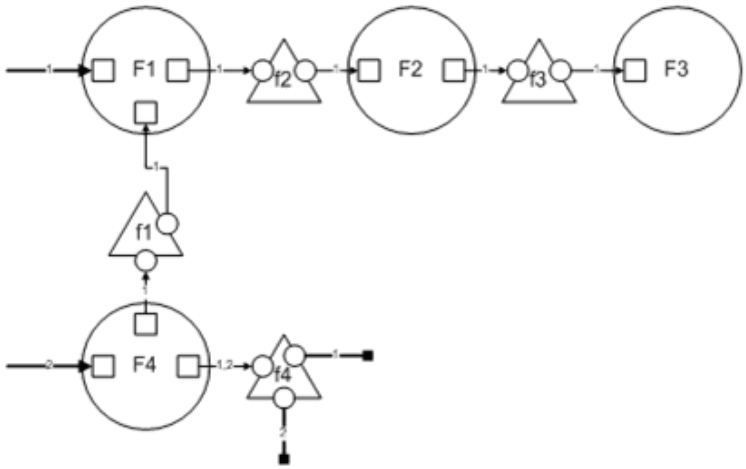

Fig. 3. Example diagram

Thus, two forms: F4 and on of the forms F1,F2 or F3 are represented on the user screen. User has simultaneous access to these forms.

The represented diagram is divided in two ones, corresponding to threads 1 and 2 (pict.4, 5).

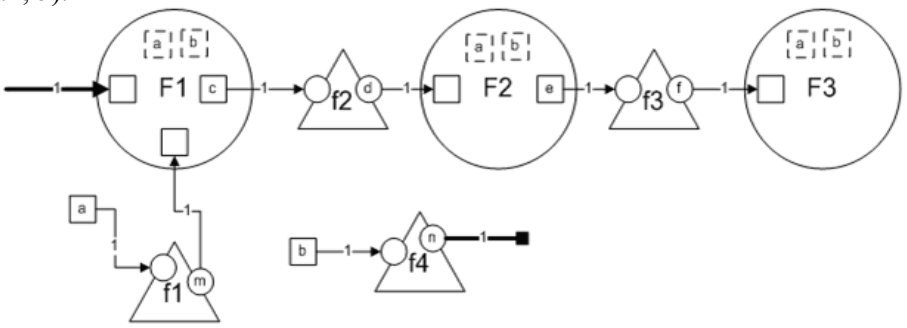

Fig. 4. Diagram with thread 1

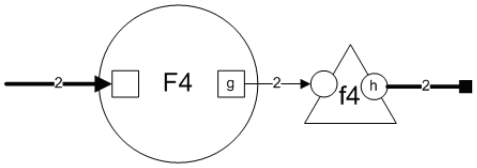

Fig. 5. Diagram with thread 2

Diagram part, describing first thread functioning, is represented on pict.4. User's activities and function activities results are represented with small letters of Roman alphabet.

FFD translation to FSM

Toolkit "core", working with one computing thread is considered further.

Used forms and functions set is identified as $Q = \{ q_1, q_2, \ldots, q_n,\}$. Possible user impact on system and events, entailed by functions execution are identified as $\sum = \{ a_1, a_2, \ldots, a_n,\}$.

It is evidently, that diagram specifies functions of transition set, according to "core" mode and appeared impact, which is identified as $\delta(q_i,a_i)$. Thus, the diagram, working with one thread , can be represented by deterministic FSM (DFSM) [6], since the following condition is fulfilled: at any number of input impacts to FSM, there can be only one FSM state, which can appear, after FSM change its current state. The FSM can be represented as

$$A = (Q, \sum, \delta, q_s, F) \quad (4)$$

where q_s – form or function, corresponding to thread start, F – acceptable FSM states set F = Q

Diagrams (pic.4, 5) are represented as FSMs (pic.6 and 7).

Input symbols for these FSMs are user actions and event, caused by functions execution. FSM states are represented with circles, input impacts are represented with arrows.

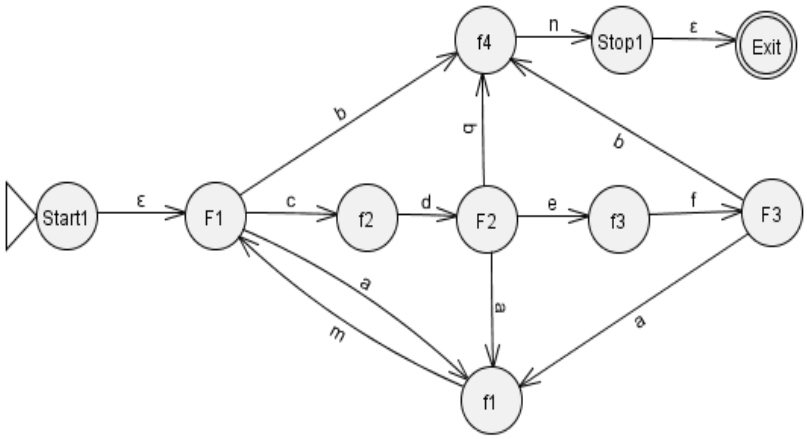

Fig. 6. DFSM

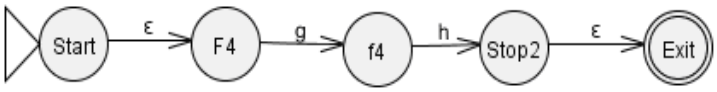

Fig. 7. DFSM 2

The impact, generated automatically, is indicated with letter ε.

Considering core functioning with a few threads, its concept should be extended.

In this case, a few forms can be represented on the user's screen, and *core* can handle a few functions simultaneously. Thus, the function of transition can be represented as

$$\{q_i\} = U\ \delta(q_i,a_i),\ \text{where } q_i \in \{ q_j \} \quad (5)$$

Thus, forms and functions diagram can be represented as non-deterministic FSM (NDFSM).

For joining a few deterministic FSM ε- bridging is used.

ε- bridging of state q_i is ε- NDFSM [6] states set, which can be got from q_i via ε-transition chain. This set consists at least of one element q_i. The function, which argument is FSM state, and its value corresponds to ε- bridging, is called **eclose**. It can be defined as

$$\text{eclose}(q_i) = \{ q_i \} \cup \{\text{eclose}(q_i) \text{ if } \varepsilon\text{-transition from } q_i \text{ to } q_j \text{ exists}\} \quad (6)$$

FSM starts working when it has states set eclose(q_0). When FSM has states set $\{q_i\}$, it changes this set to another one, got after all states ε-bridging and joining sets $\delta(q_i, a)$.

$$\{q_i\}^{next} = U\ \text{eclose}(q_j), \text{ where} \quad (7)$$

q_j belongs to $U\ \delta(q_k, a)$, where q_k belongs to $\{ q_k \}^{current}$

DFSM (pic. 6, 7) can be joined to one NDFSM (pic.8) by ε-transitions.

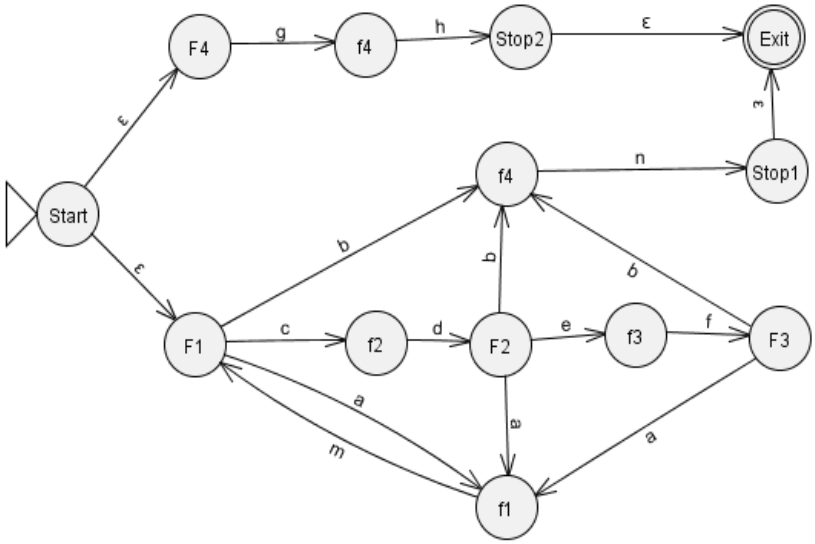

Fig. 8. NDFSM

Hence, designed system can be represented with diagram, consisting with FSMs. It allows

- to analyze system behavior at certain instants
- to optimize designed system by FSM minimization [6].

Developed diagram analysis requires input symbols generator, which can interact with user and make a decision, concerning events, caused by functions execution.

FSM functioning realization

FSM analysis, testing, and optimization require FSM functioning algorithm development and realization. Designed multithread system can be represented only by NDFSM (pic.8). FSM functioning realization would be considered in the context of NDFSM functioning realization.

NDFSM functioning by step-by-step pass

The NDFSM can be realized on deterministic computer: current states are kept, when next input symbol is handled, every state is changed to corresponding set and the results are joined.

Assuming, that sets joining is realized in constant time, independent of set size, the operations number, needed for one symbol handling can be estimated as O(M), where M is FSM states number. Usually, the complexity of sets joining linearly depends on second set size. Since it can be estimated as O(M), the complexity of one symbol handling is $O(M^2)$, and N-sized string handling complexity is $O(N*M^2)$.

ε-transition handling can be realized by signal sending before data handling and after ε symbol handling, until states set stops changing. This stage can be optimized, considering that set $eclose(q_i)$ is constant. It allows to find $eclose(q_i)$ value only one time. This stage complexity is $O(M^2)$.

This way of NDFSM functioning realization requires a lot of computing resources. It is used for complex designed systems analysis.

NDFSM functioning by its transformation to DFSM

May there is some NDFSM with three constants: q_0, q_1, q_2. This NDFSM can stay in any of the following states sets, independently of its internal structure:

- Ø (empty set)
- {q0}
- {q1}
- {q2}
- {q0, q1}
- {q0, q2}
- {q1, q2}
- {q0, q1, q2}

Summarizing features of NDFSM with any states number, DFSM can be defined as

$$q_j^{DFSM} = \{q_i^{NDFSM}\} \quad (8)$$

Input symbols alphabet is equal to NDSFM one.

DFSM function of transition can be represented as

$$\delta^{DFSM}(q_j^{DFSM}, a) = U\ \delta^{NDFSM}(q_i^{NDFSM}, a), \text{ where } q_i^{NDFSM} \text{ belongs to } q_j^{DFSM} \quad (9)$$

Primary DFSM state is the set, only consisting of NDFSM primary states.

$$q_0^{DFSM} = \{q_0^{NDFSM}\} \quad (10)$$

The idea of NDFSM transformation to DFSM based on that subsets set of state finite set is finite, i.e. independently of NDFSM behavior, it always stays in one of the states finite set.

For ε-NDFSM transition, (1) can be rewritten as

$$\delta^{NDFSM}(q_j^{DFSM}, a) = U\ eclose(q_i^{NDFSM}), \text{ where} \quad (11)$$

q_i^{NDFSM} belongs to $U\ \delta^{NDFSM}(q_k^{NDFSM}, a)$

q_k^{NDFSM} belongs to q_j^{DFSM}

DFSM primary state is ε-bridging of ε-NDFSM primary state.

$$q_0^{DFSM} = eclose(q_0^{NDFSM}) \quad (12)$$

Theoretical DFSM states number is estimated. If NDFSM has M states, then the NDFSM states are all subsets of the set $\{q_0, \ldots q_{M-1}\}$. Every q_i can belong or not belong to subset. Thus, there is 2^M DFSM states.

There are two approaches to DFSM generation.

The first approach - all the required DFSM states are generated, connections between them are fixed, then all unattainable states are removed. The second approach - only attainable states are generated at first. The last approach is considered more carefully.

Start is DFSM primary state eclose(q_0). It is attainable by its definition. If the state is attainable, than all the states, that can be changed to this one are also attainable. The state q_i can be changed to that states, which are $\delta(q_i, x)$, where δ is the function of DFSM transition, and x belongs to input symbols set. Having sorted out all input symbols, the states ($\delta(q_i, x)$) are got. The same principle is applied to all got states while their generation.

Operation number linearly depends on DFSM states number, which doesn't exceed 2^M. This approach advantage is that after DFSM is generated, it handles any symbol in constant instant, and N-length string – in O(N).

The first approach realization requires more computing operations (accordingly more time) for one step computing and less main memory. When second approach application, computing resources for one step computing have constant value.

Choice of NDFSM functioning realization way depends on:

- developed FSM complexity (designed system complexity)
- computing resources or main memory priority.

Conclusion

Suggested approach to design realization while IS development allows to raise:

- developed systems quality by early testing and optimization realization;
- IS successful development probability by raising design stage quality;
- IS general efficiency, while its functioning in heterogeneous global environment by implementing the toolkit, supporting parallel computing threads execution.

Total efficiency increase of the information system development and implementation by Information system (IS) development and reengineering process reorganization.

REFERENCES

1. F.I. Andon Foundation of Software Systems Quality Engineering. – K.: Akademperiodika, 2002. – 504 p.
2. L. Globa, A. Luntovskyy, D. Gütter, T. Kot CASE Tools for IT-System Integration // Polish J. of Environ. Stud. Vol. 16, No. 5B (2007), 148-153
3. L. Globa, CASE Tools for Distributed IT-System Accounting Multithreading // 4. Polish J. of Environ. Stud. Vol. 16, No. 5B (2007), 135-140
4. Globa L.S., Chekmez A. V., Kot T. N. Web-system interface prototype designing // Crimico'06, Sevastopol, Ukraine
5. L.S. Globa, Prof., Dr.Sci.Tech, Approaches and technologies of creating data-processing resources in the telecommunication environment, Electronics and Communication, p.2, 2005, p.17-24-29.
6. M.V. Mozgovoy, Programming Classic: algorithms, languages, state machines, compilers. Practical approach. – SPb: Nauka i Technika, 2006. – 320 p.

Надійшла 24.09.2008.