Stable Structures for Distributed Applications

Ion IVAN
Academy of Economic Studies, Bucharest, România
ionivan@ase.ro
Eugen DUMITRAŞCU
Faculty of Automation, Computers and Electronics, Craiova, România
eugen.dumitrascu@cs.ucv.ro

For distributed applications, we define the linear, tree and graph structure types with different variants and modalities to aggregate them. The distributed applications have assigned structures that through their characteristics influence the costs of stages for developing cycle and the costs for exploitation, transferred to each user. We also present the quality characteristics of a structure for a stable application, which is focused on stability characteristic. For that characteristic we define the estimated measure indicators for a level. The influence of the factors of stability and the ways for increasing it are thus identified, and at the same time the costs of development stages, the costs of usage and the costs of maintenance to be keep on between limits that assure the global efficiency of application. It is presented the base aspects for distributed applications: definition, peculiarities and importance. The aspects for the development cycle of distributed application are detailed. In this article, we alongside give the mechanisms for building the defined structures and analyze the complexity of the defined structures for a distributed application of a virtual store.

1 Distributed applications

A distributed application or a global application implies the access of data from many nodes of a computers network. The components are being executed on different nodes, on different platforms that are connected to the network.

The distributed applications are those in which many beneficiaries or users that are in different points of territory, access definite resources for computer network to solve a problem. The modern conceptions for banking transactions, inter-banking transactions, realization of e-commerce activities, training activities, informing activities, testing of knowledge activities, concluded the on-line contracts, these are just few of the distributed applications that must characterize the information society. Development philosophies for e-learning, e-government, e-business, for virtual organizations and the implementation of new work forms are based on the principles of distributed applications IVAN06].

Each user has own set of information that he has access. There is information that all users have access, but there exists information that only certain users have access. The access is done through authentication user by way a user name and a password.

The particularities of distributed applications are:

- strong interfaces that permit using them by a very diverse number of citizens;
- high generality degree that permits large number of persons to solve their own problems;
- friendly interfaces that permit the elimination of input data errors and the abandon of utilization;
- levels of security which guarantee that the system of transactions is operational;
- levels of access that convenient resolve the problem of security with the problem of transparency;
- high level of correctness and reliability;
- the guarantee for recording sufficient information that give the possibility to reconstitute the information route;
- the components of any distributed application contain two important parts: application part and communication part; some components contain a special part named administrative part with control role and manage role of components;

• high degree of modularity and the possibility of extensibility through addition or elimination of some software or hardware components;
• the possibility of many more users to share the resources;
• a large availability in case of fault of some components;
• fault tolerance.

The term *distributed application* has three aspects:
• the application A, whose functionality is divided in *n* components, A_1, A_2, …, A_n, $n \in N$, $n>1$ that interact and cooperate together; each component is a distributed application or a process;
• the components A_i are autonomous entities that run on different computers;
• the components A_i change information through network.

The importance of distributed applications is bigger and bigger in the information society. In these days almost any application is realized distributable.
A distributed application contains the procedures $PROC_1$, $PROC_2$, ….$PROC_{pr}$. Each procedure is characterized by entry points and exit points. The procedure is called from a single point or from a multiple points. The procedure with many exit points has many *return* instructions.
There are programming languages that defined many entry points for the procedure. We present all the situations in which a procedure appears with entry points, with exit points and calls between procedures.
A procedure with w parameter has format $PROC(p_1, p_2, p_3, …, p_w)$.

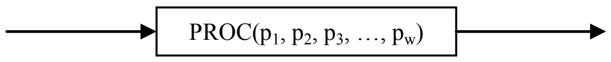

Fig.1. Procedure with one entry point and one exit point

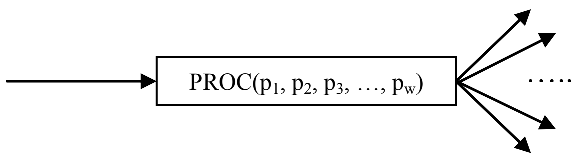

Fig.2. Procedure with one entry point and many exit points

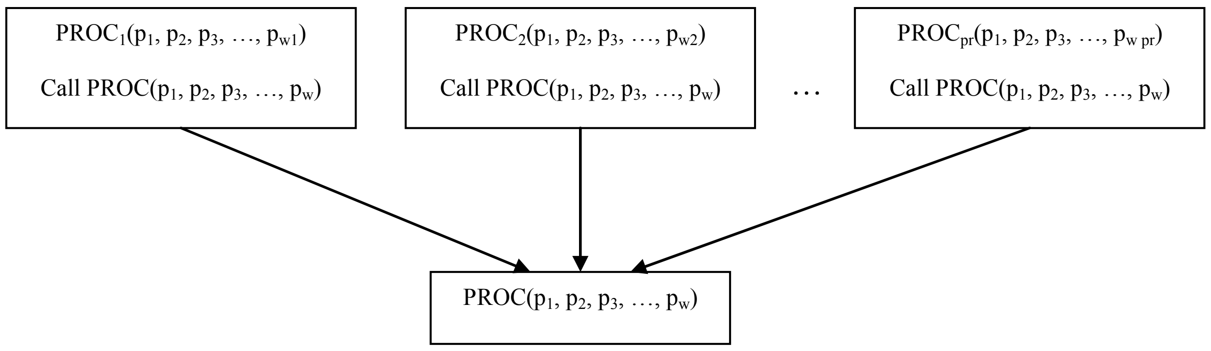
PROC1(p1, p2, p3, …, pw1)
Call PROC(p1, p2, p3, …, pw)
PROC2(p1, p2, p3, …, pw2)
Call PROC(p1, p2, p3, …, pw)
…
PROCpr(p1, p2, p3, …, pw pr)
Call PROC(p1, p2, p3, …, pw)
PROC(p1, p2, p3, …, pw)

Fig.3. Procedures that calls same procedure

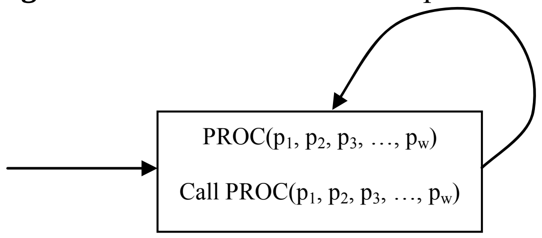

Fig.4. Procedure with one entry point and auto call – recursive procedure

Starting from the procedure idea and call, the procedures are nodes into a graph and the realized calls are arcs between nodes. Let's consider two procedures $PROC_i$ and $PROC_j$; the graph structure of call a procedure $PROC_j$ by $PROC_i$ is presented in Figure 5.
Each structure has the following features:
• homogeneity – all structures have the same number of incidence arcs towards the interior and the same number of incidence

arcs towards the exterior;
- node complexity – each node has its own complexity;
- hierarchism – the organization of structure by levels;
- the flow between nodes is characterized by the intensity or the reference frequency.

The quality characteristics of the structure are:

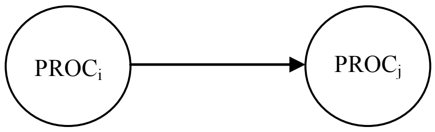

Fig.5. The call of procedure $PROC_j$ by procedure $PROC_i$

- accessibility – all nodes are referred through an access procedure for a certain rule, like for a tree structure the access of a tree in an inorder way– left-root-right –, in preorder – root-left-right –, in post order – left-right-root;
- testability – the capability to view the behavior of the element from a structure in different entries;
- interchangeability – the capability to substitute one node with another that realized at least the functions of a replaced node;
- finitude – the number of the references that belong to a closed acceptable interval, time limitation of a execute period (duration);
- correctness – the capability of the specified entry data to obtain the given results;
- generality – the potential to take over any combination of input data that belong to an interval. If $n \in [10, 20]$ and $x_i \in [1, 100]$, the program for computing the average must offer correct results for series with however much terms between 10 and 20 and for any values between 1 and 100;
- determinism – for the same input data, each time we process we obtain same results;
- stability – the potential to react of a structure: for short variation of inputs it is obtained short variation of outputs, for big variation of inputs it is obtained big variation of outputs; the potential of a structure to keep the elements on a same levels and to keep the connections (links) between nodes through: structural stability – the positions of elements and links and the behavior stability – little inputs little outputs;
- accuracy – to hint the error missing or to incorporate them in acceptable limits specified in the quality standards;
- consistency – represents the degree of inputs and outputs data that satisfy a set of conditions; we take into consideration the absence of variation and contradiction;
- correlation – the characteristic of input data or output data to be reported or compared with similar data;
- relevance – maintenance of nodes into a state that offer the capability to recover an information necessary to the user;
- suitability – potential of nodes as applications to satisfy the necessities of the user at the request time;
- validity – represents the potential of the applications from nodes to produce the desirable results.

1. The cycle of realizing the distributed applications

The life cycle of a distributed application includes the development of the cycle of application, the usage cycle, the maintenance cycle and the reengineering cycle.

The development cycle of a distributed application consists in the next phases [IVAN04]:
- analysis – what is built;
- designing – how is built;
- implementation – the real built;
- testing – with quality assurance;
- maintenance.

The analysis phase defines the system's requirements, in independent mode where they will be fulfilled. Here is defined the problem that the client wants to resolve through distributed application. The result of this phase is the requirements document that clearly specifies what have to be built up.

The designing phase succeeds the analysis phase, based on requirements which establish the architecture of the distributed system: the components of a distributed system, the interfaces and the behavior mode. The design document describes the implementation plan of requirements specified in previous phase.

It is identified the details for programming languages, developing environments, size of memory, operating system, algorithms, data structure, global type definitions, interfaces, etc.

The implementation phase continues building the system from zero or from assembling some pre-existing components. The team has to manage the problems regarding quality, performance, libraries and debugging. The purpose is to produce a proper system. An important problem is to eliminate the critical errors.

Testing phase consists in verifying the application as a whole, or on components and the possible errors or not concordance with customer requirements are eliminated, corrected or modified. At the testing of final version, it will be verified by the client the way of how it was put in practice his requirements and the way in which the propose functionalities are implemented into specifications.

The maintenance phase is the longest phase and it is lasted on whole cycle time of a software product, the application has to be maintain in optimal parameters, has to be supervised and in eventuality of discovering some errors, others than errors from testing phases, they have to be eliminated and corrected. Also, it is obtaining information from the final user for functionality mode and the possibilities of extending the functionalities and quality of software product.

The distributed application presumes work in a network with N workstations. At a workstation X_i woks K_i persons – users. A specialized user U_{ij} activates certain resources of application. The average time of an assisted transaction by the specialized user U_{ij} is DT_{ij}.

Taking into consideration the costs, for a transaction of a medium duration DT_{ij} we obtain a minimal cost per transaction CT_{ij}.

The total cost of utilization CTU will be:

$$CTU = \sum_{i=1}^{N} \sum_{j=1}^{K_i} CT_{ij}$$

The level of CTU is associated to the structure with which the distributed application is endowed.

If the structure is the best defined, CTU < CTU(S_h) where S_h is any structure with which the distributed application is endowed.

If the structure is chosen or built without considering the performance elements which are spread to the users, at least a structure S_r will certainly exist for which CTU > CTU(S_r).

The maintenance is the capability of a structure to assume the modifications of the problem solved by the informatics application.

There are design techniques of maintainable structure, modules that include prognostics for combination data types of inputs, that the user didn’t specify but the person who designs the specification can deduce them.

On realizing the time of a distributed application it is desired to obtain a higher level of global efficiency.

On a structure S_0 the phases E_1, E_2, …..E_n are covered from the design cycle of the distributed application. The definition of a structure is made on the beginning of the phases.

If the structure is perfect the costs C_1, C_2, …, C_n will be strictly specified for each phase. If the structure is not well or has to be modified the costs C_1+CA_1, C_2+CA_2, …C_n+CA_n will exist, where CA_1, CA_2. …, CA_n are the costs determinate by all what has to be done on distributed application because the structure was modified from S_0 to S_h, $h \leq n$.

The application is executed and we have the costs CE_1, CE_2, …CE_m, where m is the number of users.

The application enters in maintenance process. If the structure suffers modification from S_h to S_{h+i} the maintenance costs specified to the structures will appear.

The later the necessity of redefinition is identified, the higher the costs of modifying the application are.

The total cost generated by the structure of application appears like an additional cost due to the imperfection of the structure.

CGS=(CA_1 + CA_2 + … + CA_n)+ CNPU + CSM

where:

CGS – global cost of structure

CNPU – the cost of non performance of user

CSM – additional costs generated in maintenance by the modification of structure

A structure S_x is globally efficient, if for any

other structure S_y:
$CGS(S_x) < CGS(S_y)$
where:
S_y – other structure

2. Construction of defined structures

The defined structure corresponds to the concept that the structure of application is not modified once it was established. The structure presupposes components and elements between which there are many dependencies.
The types of defined structure are:
- linear structures;
- tree structure;
- graph structure.

The flows of defined structures are:
- non-oriented;
- unidimensional oriented;
- bidimensional oriented.

We defined a *linear structure* with n nodes X_1, X_2, …X_n. Each node X_i is linked with an oriented arc to node X_{i+1}, where X_i represents procedures or modules of distributed application ale with linear structure. The procedures or modules are called in cascade or series. In case a procedure or a model doesn't function the whole application is non-functional.

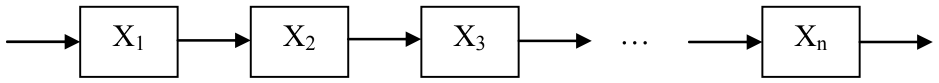

Fig.6. Linear defined structure

It is defined a *tree structure* with n nodes X_1, X_2, …X_n. Each node X_i with i>1 has a parent and is linked to an oriented arc by the node X_j, on an inferior level. The procedures or modules are called hierarchically. In case a procedure or a module doesn't function, all the descendent procedures don't function.

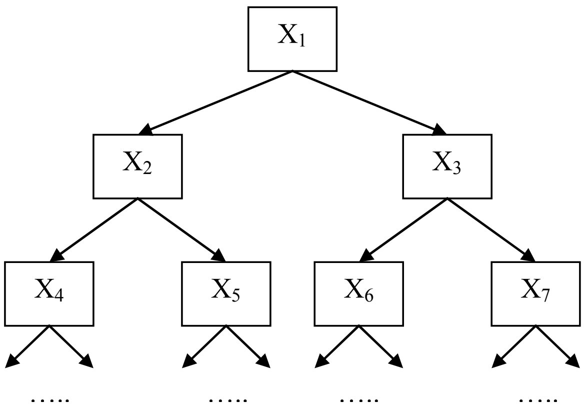

Fig.7. Tree defined structure

It is defined a *graph structure* with n nodes X_1, X_2, …X_n. Each node X_i is linked with an oriented arc by any node X_j from structure. In case the procedure or a module doesn't function, the application is still functioning if the node that does not function, doesn't determine the isolation of the other nodes, that is the ways between the nodes are keep on functioning.
The definition of defined structured application means:
- identifying the modules;
- defining the relation between modules;
- defining the type of structure;
- defining the operations between modules;

The structure is considered defined when:
- all processing functions have been defined;
- the requests of working with modules were connected to each other;
- all variables have been specified;
- all modules have been built;
- all algorithms have been established.

Redefining of structures of distributed applications means:
- adding new nodes;
- removing nodes;
- changing the position of the nodes from level k on level k+1 or from level k on level k-1;
- modifying content of a node;

• changing the position of nodes into level.

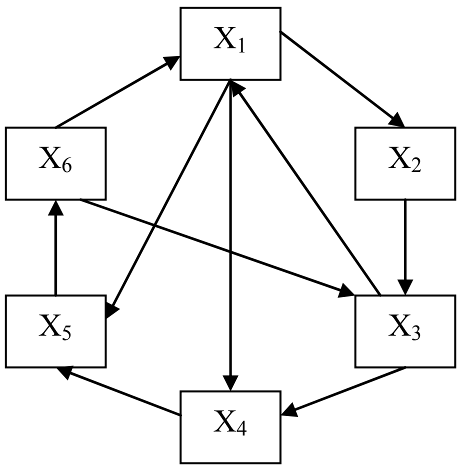

Fig.8. Graph defined structure

The probability to modify structure in phases of developing cycle is under a level that assures the stability of the structure. The defined structure is a structure with very low modified probability.

We must create premises to realize well defined structures, as being those which during the whole cycle of development are not modified. Coming back to previous phases is strictly due to the incorrect or incomplete operations that decrease the quality of the obtained components.

The well defined structure generates some detailed specification and the return is due to the differences that exist between the increase of detailed specifications and the module in the stage where the respective phase is realized. The return means the work on module. If the structure is not defined, but is a certain construction, it is worked on specifications and on module with big costs.

It is demonstrated the stability of structure using a tree structure. The complexity of the structure is computed depending on the number of ponderous nodes NRNp and the number of ponderous arcs NRAp. For a binary tree structure, the number of nodes and the number of arcs are computed depending on the number of levels for a tree niv.

The number of nodes from a level k is 2^{k-1}. The weight is given by the level that is the node, that is k. Therefore we deduce the following formula for computing weight nodes of a binary tree:

$$NRNp = \sum_{k=1}^{niv} k \cdot 2^{k-1}$$

The number of incident arcs in the nodes from level k is given by the number of nodes from that level 2^{k-1}, except of the level 1 where we have not an incident arc. We deduce the formula for computing the number of ponderous arcs:

$$NRAp = \sum_{k=2}^{niv} k \cdot 2^{k-1}$$

The Mc Cabe complexity for a binary ponderous tree structure is:

$$C=NRAp - NRNp + 2= \sum_{k=2}^{niv} k \cdot 2^{k-1} - \sum_{k=1}^{niv} k \cdot 2^{k-1} + 2 = 1$$

where:

NRNp – the number of ponderous nodes

NRAp – the number of ponderous arcs

niv – the number of levels

It is observed that for a binary tree structure that have on levels the maximum nodes, the complexity is constant and is equal to 1.

It is considered a binary tree structure with three levels like in Figure 9.

For this structure we have:

$NRNp = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 = 17$

$NRAp = 2 \cdot 2 + 3 \cdot 4 = 16$

C=NRAp-NRNp+2=16-17+2=1

If in this structure is added one node then the number of level is increased with 1 but the structure is not complete, that is it has not maximum number of nodes on level 4, so:

$NRNp = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 + 4 \cdot 1 = 21$

$NRAp = 2 \cdot 2 + 3 \cdot 4 + 4 \cdot 1 = 20$

C=NRAp-NRNp+2=20-21+2=1

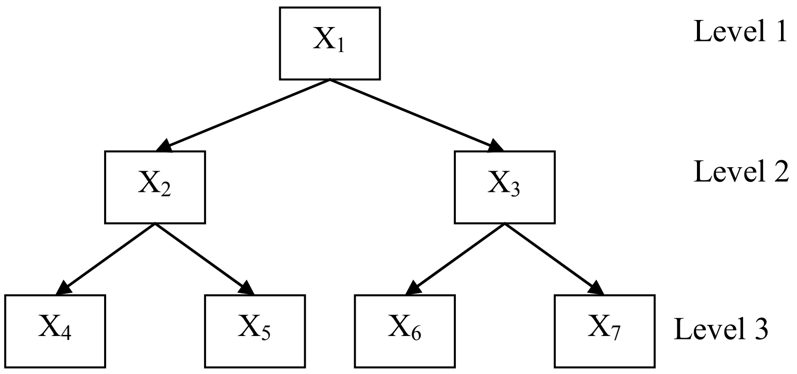

Fig.9. Binary tree structure with three levels

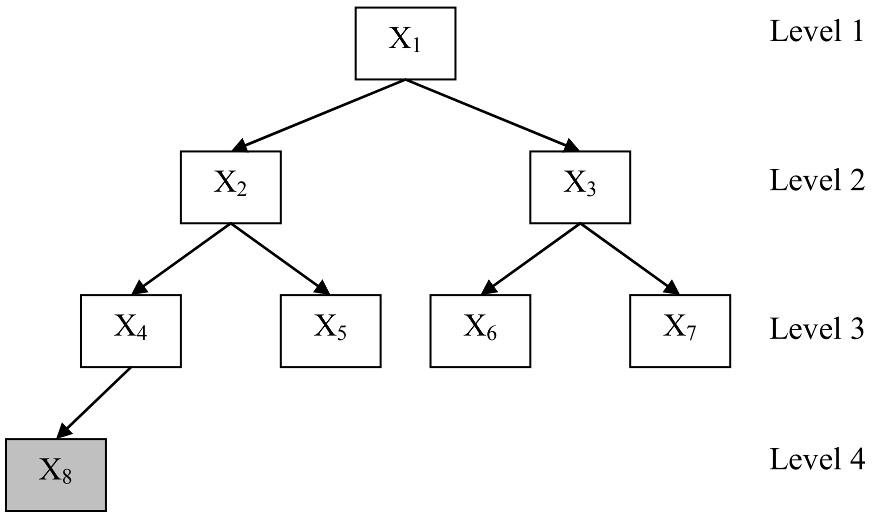

Fig.10. Adding a node in binary tree structure with three levels

It is observed that in this case the complexity remains equal to 1.

This complexity remains constant if the structure is homogeneous, and the maximum number of descendant from every node is the same.

The stability of a defined structures of distributed applications is an important characteristic. The distributed technology became every day more robust but is still immature, because optimal and safe technologies are continually experienced, which makes it easier to implement the applications. Consequently, it is necessary to know the weak points of the technologies used to correct the problems arose in order to obtain stable structures.

To simplify the process of application at the design level we have to find redundant methods, to put at good use the reusable components, to chose better algorithms for finding solutions to problem, finding standard templates for interfaces and processes, identifying the dynamic data flow, etc.

The main purpose is the stability of the defined structure through the stability of the processes for solving the problems. Often, the overtaking of memory and resources undermine the stability and the reliability of defined structure. The stability of the algorithm that is the base of the applications from structure must all be validated in this time.

The stability characteristic presupposes:

- a correct defined structure;
- a homogenous structure;
- a structure with a low or constant complexity even if the structure that has modifications;
- a reliable and robust structure;
- a structure where the modules have a high level of security;
- a fault-proof structure;
- a structure that is easy to maintain in time and where the next modifications are little or null;

• a portable structure.

The ways to increase the stability of a defined structure are:

• assuring the homogeneity of connections between modules;

• providing the nodes with the same levels of quality characteristics specific to the nodes which will result in a well aggregated product;

• creating the variant of structures and choosing the structure that corresponds to a criterion;

• testing the behavior of the structure and improving it;

• completely defining a problem and assuring a generality degree including all situations so that new elements won't appear in the problem;

• computing the structure's complexity and choosing a structure with low complexity;

• aggregating structures that are not stable to obtain a stable one.

3. The analysis of the complexity of the defined structure

The analysis of the complexity of defined structures presupposes computing the Halstead and McCabe complexity for each structure.

The Halstead complexity:

$C_H = n_1 * \log_2 n_1 + n_2 * \log_2 n_2$

where:

n_1 – number of nodes

n_2 – number of arcs

The McCabe complexity:

$C_{McC} = n_2 - n_1 + 2$

where:

n_1 – number of nodes

n_2 – number of arcs

The linear structure with n nodes from Figure 6, has m=n-1 arcs.

The Halstead complexity is:

$C_H(SL) = n * \log_2 n + m * \log_2 m = n * \log_2 n + (n-1) * \log_2 (n-1)$

where:

n – number of nodes

m – number of arcs

The McCabe complexity:

$C_{McC}(SL) = m - n + 2 = (n-1) - n + 2 = 1$

This means that the linear structure that has many components has the McCabe complexity very low that is 1.

In *tree structure* with niv levels, n nodes and m arcs, and each node has u descendents, we deduce the formula:

$$n = \frac{u^{niv} - 1}{u - 1}$$

$$m = n - 1 = \frac{u^{niv} - 1}{u - 1} - 1 = \frac{u^{niv} - u}{u - 1}$$

The Halstead complexity is :

$$C_H(SA) = n * \log_2 n + m * \log_2 m = \frac{u^{niv} - 1}{u - 1} * \log_2 \frac{u^{niv} - 1}{u - 1} + \frac{u^{niv} - u}{u - 1} * \log_2 \frac{u^{niv} - u}{u - 1}$$

where:

niv – number of level

u – number of descendents

Mc Cabe complexity:

$$C_{McC}(SA) = m - n + 2 = \frac{u^{niv} - u}{u - 1} - \frac{u^{niv} - 1}{u - 1} + 2 = 1$$

That means that the tree structure that has many components has the McCabe complexity very low that is 1.

In a complete *graph structure* with n nodes and m arcs, each node having connections with others n-1 nodes, we deduce the formula:

$$m = \frac{n \cdot (n-1)}{2}$$

Halstead complexity is:

$$C_H(SG) = n * \log_2 n + m * \log_2 m = n * \log_2 n + \frac{n \cdot (n-1)}{2} * \log_2 \frac{n \cdot (n-1)}{2}$$

where:

n – number of nodes
m – number of arcs
Mc Cabe complexity:

$$C_{McC}(SG)=m - n + 2 = \frac{n\cdot(n-1)}{2} - n + 2 = \frac{n\cdot(n-3)}{2} + 2$$

The graph structure that has many components has the Mc Cabe complexity very high, depending on the number of nodes.
Next, we will present the aggregations of structures, computing the McCabe complexity for them. Let's define the aggregate operator o.
The types of aggregations on structures are:
a) The aggregation of a linear structure lead to a linear structure

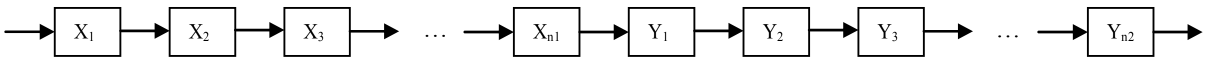

Fig.11. Aggregation of a linear structure

Let's consider two linear structures SL_1 with n1 nodes and SL_2 with n2 nodes. The last node from structure SL_1 is linked with the first node from structure SL_2.
$C_{McC}(SL_1)= 1$
$C_{McC}(SL_2)= 1$
$C_{McC}(SL_1\ o\ SL_2)= ((n1- 1)+(n2 - 1) +1) - (n1+n2) +2=1$
Therefore $C_{McC}(SL_1\ o\ SL_2) \neq C_{McC}(SL_1)+ C_{McC}(SL_2)$

b) The aggregation of a tree structure lead to a tree structure

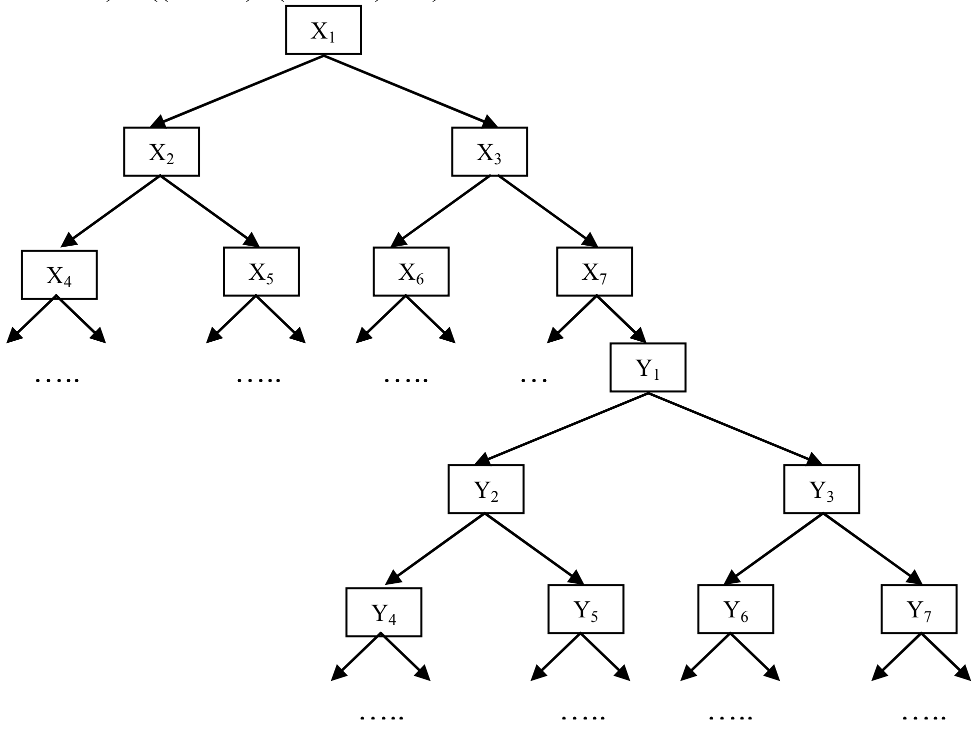

Fig.12. Aggregation of an unbalanced tree structure

Let's consider two tree structures SA_1 with n1 nodes and niv1 levels and SA_2 with n2 nodes and niv2 levels. A leaf node from structure SA_1 is linked with the root of the structure SA_2.
$C_{McC}(SA_1)=1$
$C_{McC}(SA_2)=1$
$C_{McC}(SA_1\ o\ SA_2)= (n1-1+n2-1)-(n1+n2)+2=0$

For realizing this aggregation, less balanced than the previous one, we link the roots of the two tree structures SA_1 and SA_2 as descendents of a new node R that became root of an aggregate structure. The aggregate structure contains now n1+n2+1 nodes and n1-1+n2-1+2 arcs.

Then the complexity of aggregation will be:

$C_{McC}(SA_1\ o\ SA_2)=$ (n1-1+n2-1+2)-(n1+n2+1)+2=1

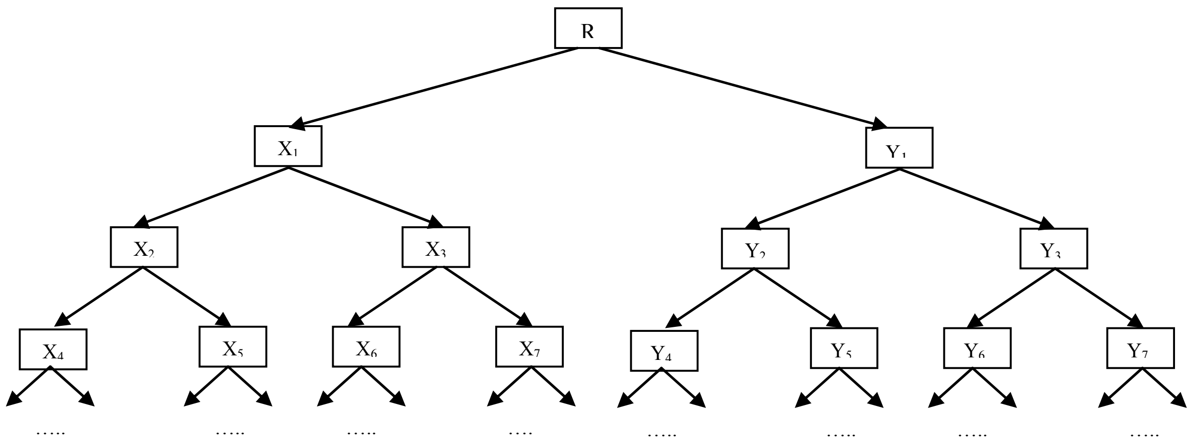

Fig.13. Aggregation of a balance tree structure

c) The aggregation of graph structures leads to a graph structure

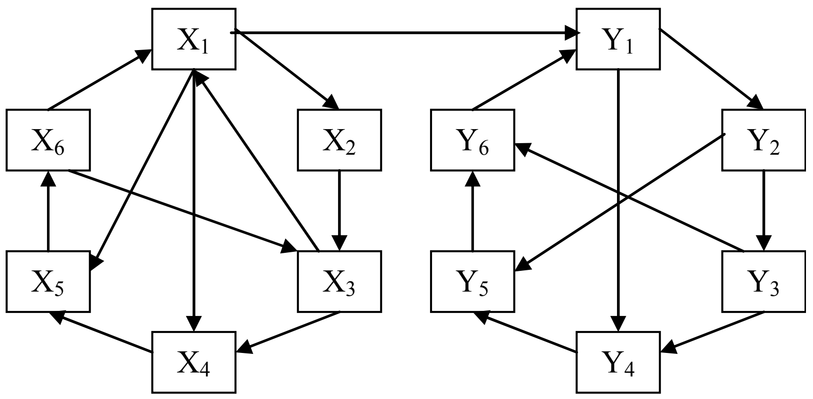

Fig.14. Aggregation of a graph structure

Let's consider two graph structures SG_1 with n1 nodes and SG_2 with n2 nodes. A node from structure SG_1 is linked to a node from structure SG_2.

$C_{McC}(SG_1)= \frac{n1\cdot(n1-3)}{2}+2$

$C_{McC}(SG_2)= \frac{n2\cdot(n2-3)}{2}+2$

$C_{McC}(SG_1\ o\ SG_2)=$

$(\frac{n1\cdot(n1-1)}{2}+\frac{n2\cdot(n2-1)}{2})$ - (n1+n2)+2= $\frac{n1\cdot(n1-3)}{2}$ + $\frac{n2\cdot(n2-3)}{2}$+2

d) The aggregation of a linear structure with tree structure leads to a tree structure

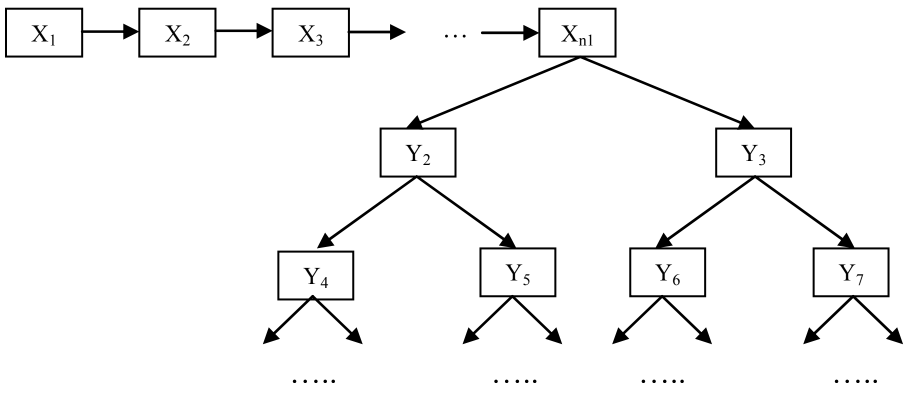

Fig.15. Aggregation of a linear structure with tree structure

Let's consider a linear structure SL_1 with n1 nodes and a tree structure SA_2 with n2 nodes. The last node from structure SL_1 is root for structure SA_2.

$C_{McC}(SL_1)= 1$

$C_{McC}(SA_2)=1$

$C_{McC}(SL_1 \ o \ SA_2)= (n1-1+n2-1) – (n1+n2-1) + 2=1$

Deci $C_{McC}(SL_1 \ o \ SA_2) \neq C_{McC}(SL_1)+ C_{McC}(SA_2)$

e) The aggregation of a linear structure with graph structure leads to a graph structure

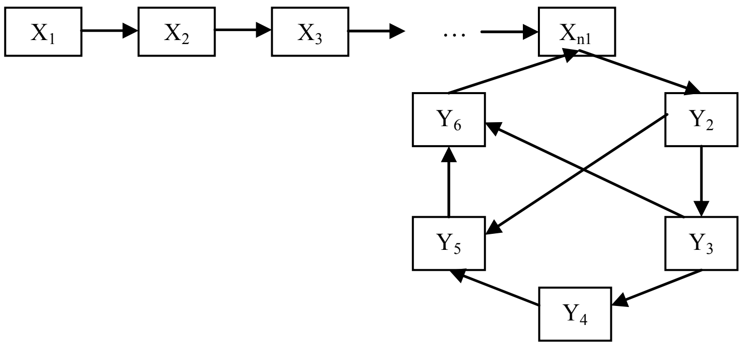

Fig.16. Aggregation of a linear structure with tree structure

Let's consider a linear structure SL_1 with n1 nodes and a graph structure SG_2 with n2 nodes. The last node from structure SL_1 is node from structure SG_2.

$C_{McC}(SL_1)= 1$

$C_{McC}(SG_2)= \frac{n2 \cdot (n2-3)}{2}+2$

$C_{McC}(SL_1 \ o \ SG_2)= (n1-1+\frac{n2 \cdot (n2-1)}{2}) – (n1+n2-1) + 2=\frac{n2 \cdot (n2-3)}{2}+2$

Therefore $C_{McC}(SL_1 \ o \ SG_2)= C_{McC}(SG_2)$

f) The aggregation of tree structure with graph structure leads to a graph structure

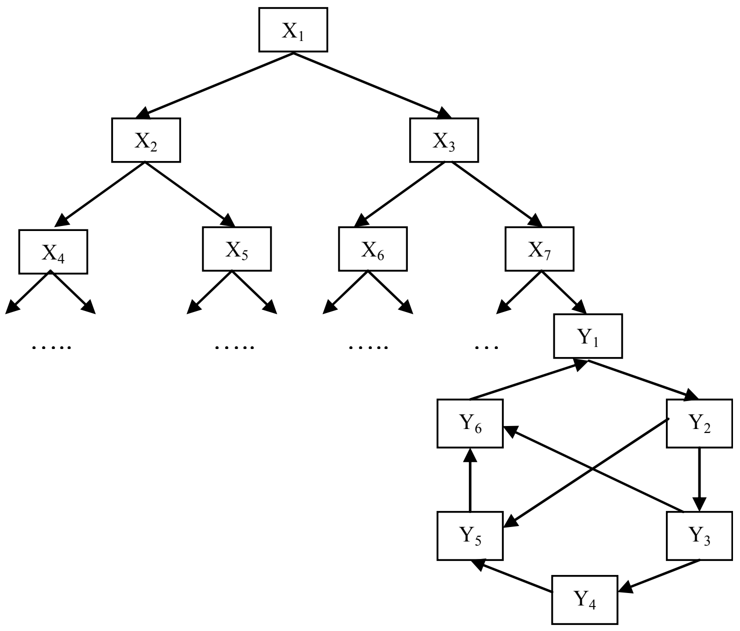

Fig.17. Aggregation of a tree structure with graph structure

Let's consider a tree structure SA_1 with n1 nodes and a graph structure SG_2 with n2 nodes. One leaf node from structure SA_1 is linked to a node from structure SG_2.

$C_{McC}(SA_1)= 1$

$C_{McC}(SG_2)= \frac{n2 \cdot (n2-3)}{2}+2$

$C_{McC}(SA_1\ o\ SG_2)= (n1-1+\frac{n2 \cdot (n2-1)}{2}+1) - (n1+n2) + 2= \frac{n2 \cdot (n2-3)}{2}+2$

So $C_{McC}(SA_1\ o\ SG_2)= C_{McC}(SG_2)$

4. The defined structure for a virtual store

The applications in virtual stores are web applications for presenting and e-commerce. The client analyses the offer in an online store by accessing the pages from the store's site, chooses the desired products which he stores into a *virtual basket* and then when he is determined to buy them, he sends a request to start the electronically transaction for buying. The payment is done by electronic card. The client provides the personal data and information regarding his card to a server S_1 that authenticates the information sent by the client and it makes the connection with the account from the bank where he has card, from where he will pay the products to the store or product company. On this server there is an application of the type *payment gateway* that represents an interface between operations from banking network and electronic transaction. This application with transaction for buying is named *Point Of Sale* - POS.

All information relating the products to be bought, regarding customer's requests, regarding the information sent by the customer, are stored into a database located on the web server. The administration of the supplies and the update of the database are done by the application server named S.

After the validation and authentication of customer's card has been done, the server S sends the requests of the customer to the server S_2, that makes the link with the producers of goods and it places the request of customer to them. They do the delivery of goods at customer domicile directly, by mail, or through other delivery mode. The server S_2 communicates with server application S which sends the confirmation that the products were delivered and then updates the supply from database.

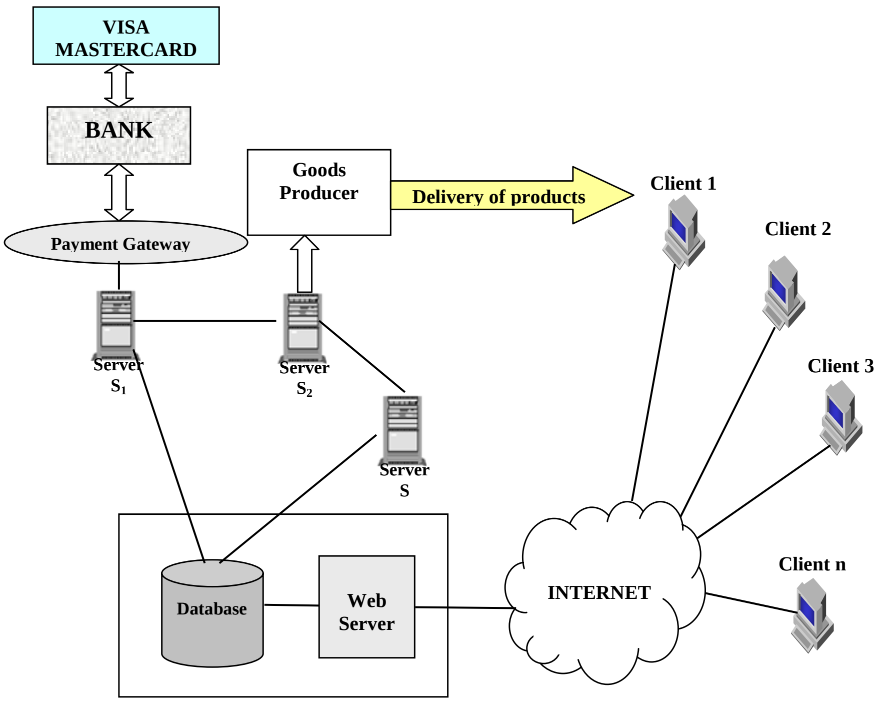

Fig.18. The structure of a virtual online store application

The web applications such as a virtual store have tree structure. For choosing a product, the user has to cover a complete way in the tree from the root to the leaf nodes [IVAL05].
An important aspect, that has to be take in account to design such a kind of application, is that the application has the facilities to auto-organize in dynamic mode depending on the reference frequencies of corresponding pages [IVAL05].
The quality characteristics of an application like virtual store according to [IVAN07] are:

- functionality – the ability to satisfy the needs and requirements of the user; the important sub-characteristics of this characteristic are:
  - completeness - referring to the degree where the application comprises the necessary and sufficient functions for satisfying the user requirements;
  - correctness – specifies the degree where the results are closer than real results;
  - security – the quality to assure the data security against data losses and non-authorizing access;
  - compatibility – the degree where the application is implemented without modify existing major software;
  - interoperability – the capability to communicate with other applications.
- reliability – capability of application to maintain the performance level in stable conditions, on a stable period of time; the sub-characteristics of this are:
  - fault detection – the degree where the application continues to operate in spite of the errors without affecting the user;
  - availability – the degree where the product is operational taking into account the flaw of the hardware or software system;
  - recoverability – refers to re-establishing the level of performance and the recovery of data direct affected with specify efforts when fault of system.
- usability – is the set of attributes that permits the effort specified by using from a set of users or implicit users:
  - easy to use – represents the effort of user to understand and use the application;
  - operability – refers to operating and controlling of the operation from the users.
- efficiency – the set of attributes that allows keeping the relation between the level of software performances and the level of using resources in special stable conditions; the sub-characteristics of this are:
  - time's economy – the application capacity to process data and to offer results in optimal intervals of time;

- resource's economy – the capacity of application to offer final situations using limited software and hardware resources, when the quality rapport results/cost are to be optimal.

• maintainability – the set of attributes that refers to the necessary effort for doing certain modifications; the modifications include corrections, the adaptation of application in a context of change and the modifications of specification requirements and functionality; the sub-characteristics of this are:

- correction – refers to the effort for correcting the software errors and for meeting the user requirements;
- development – the necessary of resources to improve the application.

• portability – the set of attributes that assure the capability of application to transfer from a context to another; the context includes organizational, hardware and software context; the sub-characteristics of this are:

- hardware independence – the degree where the application does not depend on specific hardware environment;
- software independence – the degree where the application does not depend on specific hardware environment, for example operating systems;
- reusability – the degree where the software is reusable in products differently, such as the initial product.

We consider a virtual store application for presenting and marketing IT products, as in Figure 19. For choosing a certain computer, the user starts from the root to the leaf nodes. The user chooses initially the Computer section, then the type of computer like Desktop or Notebook, then the producer Dell or HP, and finally the type of processor of computer.

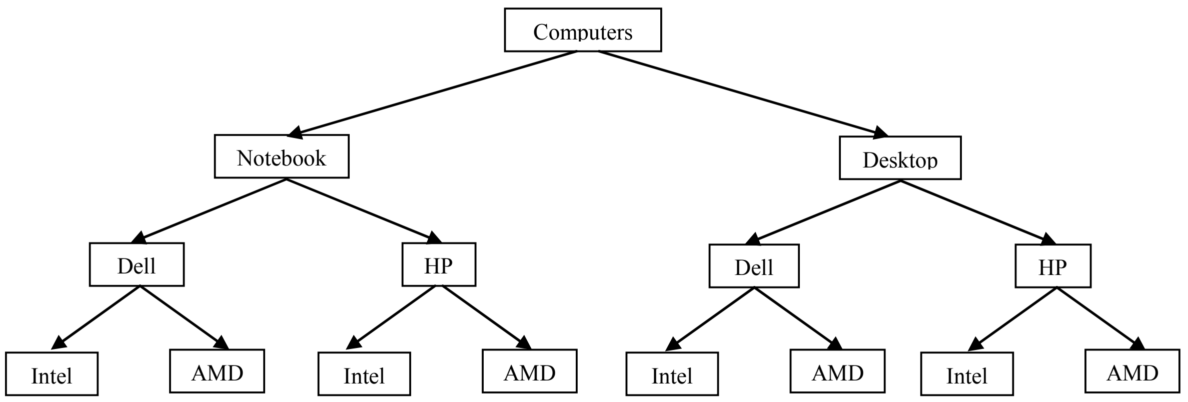

Fig.19. The initial organization of a tree structure for an online virtual store application

If the application quantifies the frequency of reference for web page, when, after a while, it is observed that the maximum frequency is associated with the producer, then the tree structure will be re-organized in a dynamic mode in order to reflect the user's preferences, as in Figure 20.

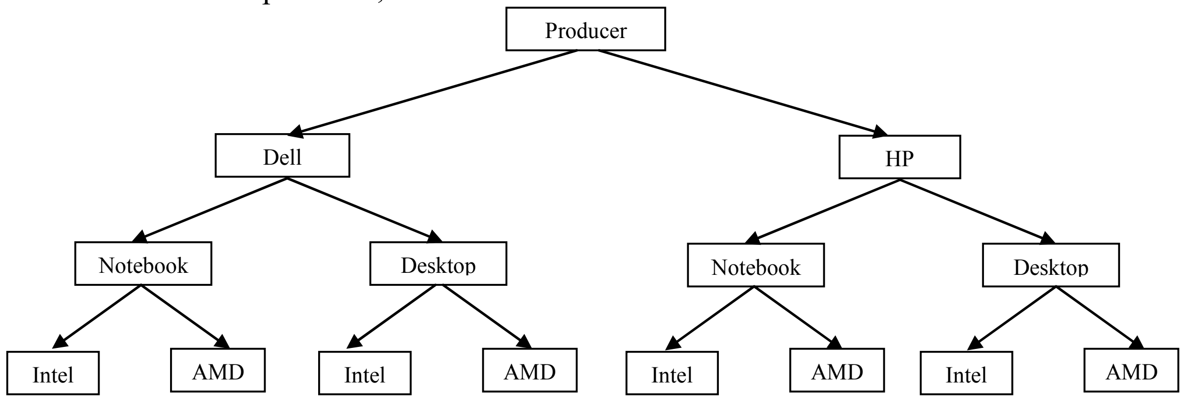

Fig.20. Re-organization of the tree structure for an online virtual store application depending on the frequencies of reference

In elaborating this type of application we must try to balance the level of complexity on structure topology in order for the web structure to be more homogenous from the referencing point of view.

The McCabe complexity of this application is:

C_{McC}=14 – 15 +2 = 1

5. Conclusions

For realizing a stable structure, even from the defining phase of the problem, we have to clarify all the elements so that the specifications obtained to incorporate the following points:

- data input types must be completely defined;
- the relation between data types must be completely defined and oriented;
- the computed and selection models must be completely and correctly defined;
- the aggregated date must be complete, correct and consistently defined.

When analyzing the problem, the sub-definition must be avoided by missing input data, computed data and aggregated variable, and supra-definition through inserts of relations, the redundant data input, or inconsistent generator.

If we consider the distributed applications AD_1, AD_2, …AD_r that proves the global efficiency, we analyze their structures, and create the structure model. At the time when a new distributed application is necessary, we try to completely or partially reuse a structure from the set of model structures.

In this context, at the level of each organization we must develop software to build databases automatically populated with levels of structure and behavior indicators of distributed applications that are in diverse phases of elaboration cycle.

In [IVAN05] we develop the software that uses the containing of databases, adding only models that defined the indicators for analysis of quality characteristics for distributed application.

At the same time, validating the models also takes place, for the stability of the structure and for analyzing the degree of defining the structure of the distributed application.

We consider the distributed applications AD_1, AD_2, …AD_r, taken at the level of estimation of a characteristic C_j using indicator Q_j. The effective level of characteristic C_j is taken, using the same indicator Q_j. When the differences between the estimated levels and the effective levels are insignificant, represented an important weight, we consider the indicator Q_j validated by practice. In the same way, we will do the same for all the quality characteristics of the distributed application.

Bibliography

[IVAL05] Ion IVAN, Felician ALECU, *Structuri HTML,* Editura ASE, Bucureşti, 2005

[IVAM06] Ion IVAN, Cristian AMANCEI, *Stabilitatea coeficientilor modelului global de calitate software*, Editura ASE, Bucureşti, 2006

[IVAN04] Ion IVAN, Cosmin IVAN, Mihai MARINESCU, *Calibrarea aplicaţiilor informatice*, Revista Română de Informatică şi Automatică, vol. 14, nr. 1, 2004

[IVAN05] Ion IVAN, Adrian VISOIU, *Baza de modele economice,* Editura ASE, Bucureşti, 2005, ISBN 973-594-571-1

[IVAN06] Ion IVAN, Eugen DUMITRAŞCU, Marius POPA, *Evaluating the Effects of the Optimization on the Quality of Distributed Applications,* Economic Computation and Economic Cybernetics Studies and Research, vol. 40, nr. 3-4, 2006, pg. 73 - 85, ISSN 0424-267X.

[IVAN07] Ion IVAN, Cătălin BOJA, *Practica optimizării aplicaţiilor informatice*, Editura ASE, Bucureşti, 2007, ISBN 978-973-594-932-7

[IVBO05] Ion IVAN, Catalin BOJA, *Managementul calităţii proiectelor TIC*, Editura ASE , Bucureşti, 2005, ISBN 973-594-558-4.

[IVPO05] Ion IVAN, Marius POPA, Entităţi text, dezvoltare, evaluare, analiză, Editura ASE, Bucureşti, 2005, ISBN 973-594-663-7