HYPERTEXT ENTITIES' SEMANTIC WEB-ORIENTED REENGINEERING

Cosmin TOMOZEI[1]
PhD Candidate, University Assistant
Mathematics and Computer Science Department
University of Bacau, Romania

E-mail: cosmin.tomozei@ub.ro

Abstract: *This paper's aim is to define the concept of Hypertext Semantic Web-Oriented Reengineering (HSR) as a process of distributed applications development which takes into consideration the semantic aspect in information retrieval and communication. It is virtually possible to apply the reengineering on web applications concerned being about the efficiency of the ideas of data structures and implementation than to mainly being troubled with the language or syntactic point of view. This research also brings some examples of distributed applications types, some small segments of them being mainly explained as well, in order to make our theory strongly connected with the practical work from software companies.*
It is very important that semantic approaches to be implemented while developing software applications, mostly when reengineering is integrated in the development process, as a step for the evolution to the next generation of web.

Key words: *distributed systems; distributed applications; web semantic; reengineering; web service; interoperability; distributed databases; dependable applications*

1. Semantic Reengineering

Reengineering appears as a result of the necessity to improve the quality of the results of one initial entity, such as software or text in order to realize new objectives which do not differ significantly from the old ones. Reengineering also may be formalized by a function, that may be simply called reengineering function which consists of the amount of methodologies, methods, procedures and techniques that applied to a software entity makes possible the transformation and integration of the new semantic principles.

This function depends of the modules that appear as variables or arguments initially, after that reengineering appearing as a product of the initial function, which is called development function in the initial moment (1). The development function from the stage 1 includes the function of reengineering.

$$Dev(\bigcup_{i=0}^{n-1} Mo_i) = \sum_{i=1}^{t} Obj_i = Obj \qquad (1)$$

The development process in this case appears as a function which depends of the modules of the software application. Semantic aspects are not taken into consideration that means the modules are only considered to be semantically insignificant. This point of view is very efficient and easy but reductionist, because each module of a software application is enhanced with semantic.

If we consider the semantic aspects of computer science and their implication in distributed software applications, semantic web will appear as a new paradigm that implies changes in our thinking about web development.

The product of reengineering and development function brings the possibility of having realized the new objectives. Reengineering functions have a variable number of arguments, each argument becoming a type of reengineering. If the semantic aspect has the main proportion in this function, then semantic reengineering is created or the main share of reengineering is the semantic one.

$$\mathrm{Re}\,eng(\arg_1 ... \arg_n) * Dev(\bigcup_{i=0}^{n-1} Mo_i) = \sum_i Obj_i^{\ 1} = Obj^1 \qquad (2)$$

The above equation (2) describes synthetically the process of reengineering due to the changing of the application's objective. Each software product has one main objective that appears as a sum or reunion of smallest objectives. Changing or modifying the objective will affect the entire thinking about the existent entity.

Objective's modification may appear as a quality driven process that will imply reengineering in order to grow the quality of the software. It will be a lot easier to measure the quality of each module if decided to hold the objectives and predictive results in a table of association. We consider, for the straightforwardness' sake that each module only has one objective as a segment of the main objective. The realization is measured by a numeric scale from 1 to 100, but other ways to measure it numerically are also accepted.

Table 1. The correspondence between objectives and modules

Module / ***Objective***	*Objective* $_1$	*Objective* $_2$	.....	*Objective* $_n$
Module $_1$	80	0		2
Module $_2$	15	76		0
.....				
Module $_n$	5	14		55
Realization	**100**	**90**		**57**

Semantic reengineering is not a standalone process. It does not exclude the other arguments and needs them to complete the tasks. Language semantics is very important in information exchange as well as data semantics and code semantics. The main idea is to develop structural and architectural transformations with minimum efforts with the result of restructuring the entities' architecture, design and source code that will bring evolution in the meaning, in contrast with the syntactic point of view. We would like to show that syntax must subordinate itself to semantics, because *the meaning* is the most important. Syntactic modification is only a way for bringing evolution in semantics.

Operational semantics consists of rigorousness of functions, procedures and programs that implement mathematic algorithms with mathematical meaning.

Reengineering in operational semantics comes as a transformation or metamorphosis of mathematic formulae in source code.

Operational semantics is stalwartly related with the transition of a software system. In [Plotkin81][2] system transition is described as being formed by a set of configurations and a binary relation which brings the system from an initial configuration (γ) to a final one (γ^1). We consider each configuration related with a meaning which is not significantly different from the previous one. Transformation from an initial to a final phase that brings a qualitative growth in the meaning and a better realization of the objectives is defined as *semantic reengineering*.

The process of semantic reengineering is finite, deterministic, value orientated and progressive. The following elements will be more explanative [TOVA08] and [IVPOTO05]:

- *finiteness* which means that reengineering is bounded limited and will be finalized after realizing all the transformations.
- *determinism*, due to the necessity of doing a number of iterations until the process is finalized and the meaning of the software faced a new qualitative upward;
- *continuity* during the development cycle;
- *flexibility* regarding the development platforms;
- *appropriate management* which guides the development team in the process.

Denotational semantics is the approach in which the programming languages meanings are used by constructing mathematical objects that describes them. Denotational semantics is very important as well, being related with the states, in which commands are partial functions of the domains of states, denotation of data types and datta structures, such as graphs, trees or vectors.

Denotational semantics of concurrency introduced new models for concurrent computation such as the actor model or Petri nets. It is as well studied from the denotational point of view the sequentiality of programs and source to source translation. For example if we consider the following web application functions it may be possible to use semantic reengineering and denotational semantics to translate it from one language to another.

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As  System.EventArgs) Handles Button1.Click
      Dim strcon As String = "Data Source=ASE-EY5776NZ4X6\SQLEXPRESS;Initial Catalog=seminarii;Integrated
Security=True"
      Dim scoma As String = "select * from firme"
      Dim sconn As SqlConnection = New SqlConnection(strcon)
      Dim ds As DataSet = New DataSet()
      sconn.ConnectionString = strcon
      Dim sq1 As SqlCommand = New SqlCommand()
      sq1.CommandText = scoma
      Dim sdadpt As SqlDataAdapter = New SqlDataAdapter(scoma, sconn)
      sdadpt.Fill(ds, "sursa")
      For Each dr As DataRow In ds.Tables("sursa").Rows
         If Convert.ToInt32(dr("id")) = 1 Then
            TextBox1.Text = Convert.ToString(dr("id_firme"))
            TextBox1.Text + Convert.ToString(dr("den_firme"))
             End If
      Next
      MsgBox("Salut")
   End Sub
```

An experienced programmer may as well transform it for reducing the number of code lines and the keeping the same functionalities. Mathematical semantic translation may be used to keep the functions and the objectives of the web application correct and good.

```
protected void Button3_Click(object sender, EventArgs e)
   {
      string strconn =
System.Configuration.ConfigurationManager.ConnectionStrings["string_config_seminar"].ConnectionString;
      SqlConnection scon = new SqlConnection(strconn);
      string str="select * from firme";
      SqlCommand scm = new SqlCommand(str, scon);
      SqlDataAdapter sadapt = new SqlDataAdapter();
      sadapt.SelectCommand = scm;
      DataSet ods = new DataSet();
      sadapt.Fill(ods,"tabel1");
      foreach (DataRow dr in ods.Tables["tabel1"].Rows)
      {
         TextBox2.Text += dr["id_firme"];
         TextBox2.Text += dr["den_firme"];

       }
  }
```

Axiomatic semantics is an approach that provides rigorousness and correctness to computer programs and is based on mathematical logic. Operators from programming languages are deeply connected with mathematical logic. If we consider as an example XOR, Logical AND, Logical OR we will see how mathematical logic affect the way of representing reality by computer programs. Axiomatic semantics has strong associations with Hoare Logic [www1], having as central element a formalism which is called Hoare Triple {P} C {Q}, where P and Q are assertions and Q is the command. It describes how a software code is changing the state of computation.

Web orientation of software applications manages to make transformations in software code and specifications from both directions, syntactic and semantic. Web services, which allow communication to take on between software programs by SOAP [www2] use web methods as traditional software applications use normal methods. Reengineering will adapt the piece of code offering it the possibility to more general and interoperable. Transformation by reengineering produces the following piece of code as web method with the result of an XML Dataset, understood by software applications which communicate over the Internet. The clients will retrieve information from the relational database just by referring and invoking web methods.

```
  [WebMethod (Description ="Citeste tot si afiseaza xml,Cosmin Tomozei")]
  public System.Data.DataSet citeste_tot()
  {
        SqlConnection scon = new
        SqlConnection(ConfigurationManager.ConnectionStrings["conn1"].ConnectionString);
        scon.Open();
        string  st =  @"SELECT intalnire.dataintalnire, intalnire.ora,intalnire.durata,intalnire.linki,
        perscon.nume + ' ' + perscon.prenume AS Invitat, Institutie.deninstitutie, utilizator.prenume + ' ' +
        utilizator.nume AS Participant, tara.codtara,Localitate.denloc,Locuri.denl FROM intalnire INNER JOIN
        perscon ON intalnire.persconid = perscon.persconid INNER JOIN Institutie ON intalnire.codinstitutie =
        Institutie.codinstitutie INNER JOIN utilizator ON intalnire.utid = utilizator.utid;
    SqlDataAdapter adpt1 = new SqlDataAdapter();
     SqlCommand scmd1 = new SqlCommand(st, scon);
```

```
    System.Data.DataSet ods1 = new System.Data.DataSet();
    adpt1.SelectCommand = scmd1;
    adpt1.Fill(ods1, "tabela_noua1");
    return ods1;
}
```

2. Semantics in Distributed Web Applications

Resource description framework (RDF) is a language which allows information to be represented and exchanged on the web. Semantic web appears as a new challenge for web developers to create applications that share data across any barriers, in a more efficient way.

RDF is based on the graph data model [www3] that uses triples formed by *subject, predicate* and *model* and represents a kind of relation between things. Efficiency appears in consequence when metadata is widely used by web applications during communication.

The syntax of this new language uses the XML syntax because is widely used in machine to machine communication over the Web. Another important aspect of the XML language is that is also human readable and understandable with very little effort. XML provides to hypertext entities more power and capacity of being machine readable.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <citeste_tot1Response xmlns="urn:ub.ro/seminarii">
      <citeste_tot1Result>
        <xsd:schema>schema</xsd:schema>xml</citeste_tot1Result>
    </citeste_tot1Response>
  </soap12:Body>
</soap12:Envelope>
```

The result after invoking the web method consists of an XML machine to machine communication file. The file serializes the dataset returned by the web method, which contains data extracted from the distributed databases, or other information that is necessary in our development process. Below is just a piece of information regarding costumers and meetings from an SQL Server Database.

```
<NewDataSet>
<tabela_noua1 diffgr:id="tabela_noua11" msdata:rowOrder="0">
<dataintalnire>25/11/2008</dataintalnire>
<ora>12:20</ora>
        <Invitat>Cosmin Tomozei</Invitat>
        <deninstitutie>CCIT</deninstitutie>
        <Participant>Popescu Pop</Participant>
        <codtara>196</codtara>
        <denloc>Johannesbourg</denloc>
        <denl>str k 1</denl>
</tabela_noua1>
<tabela_noua1 diffgr:id="tabela_noua12" msdata:rowOrder="1">
        <dataintalnire>30/11/2008</dataintalnire>
```

```
        <ora>12:20</ora>
        <Invitat>Simona Varlan</Invitat>
        <deninstitutie>CCIT</deninstitutie>
        <Participant>Popescu Pop</Participant>
        <codtara>196</codtara>
        <denloc>Johannesbourg</denloc>
        <denl>str k 1</denl>
        </tabela_noua1>
</NewDataSet>
```

RDF has the possibility of cooperating with the relational model in distributed web applications that have relational SQL or Oracle Databases, many specialists claiming that a relational view of the Semantic Web [NEWM07] can be developed due to the efficiency, openness and robustness of the relational model. We deeply agree with that idea and consider that the relational model as well as the object oriented model can be used efficiently in Semantic Web.

SPARQL is the query language for RDF, which allows retrieval of information from the graph model and has the same relational algebra query operations as in the relational model, such as *projection*, *selection* or *join* and also RDF graphs structure may be easily transformed in relational tables as vice versa. SPARQL [NEWM07] can be seen as an extension of the relational model, as we also believe.

Administration of distributed applications [EBER06] has evolved, implying semantic management. *Ontology* has to be implemented as an important component of semantic web distributed applications. Ontology represents a set of attributes or characteristics that are in a specific domain, including the relations between them. Instances (objects), classes, assertions rules and also events are components of ontology. They can be merged, generalized and also inheritance can take place in order to create an ontology which is more specific for a particular objective of a semantic web application. Ontologies formalize concepts and relations between concepts similarly to the class diagram from the UML [EBER06]. In this case they may be seen as a source for interoperability, which is an important metric for distributed applications, and make them easier to integrate in distributed systems. Due to the formalization based on computational logics they are clear and very easy to understand, not being ambiguous and unformalized.

Taxonomy contributes as a science of classifications to create ontology hierarchies and ordinate them due to our interests and their semantics. OWL is the language. The following example points out reengineering results of transformation from LISP to OWL, both languages being semantic web related.

```
(make-class student
(agent (computer_science_student))
(is-a (value computer_science_faculty)))
(instrument  (software))
(location (Computer_Lab_Corp_C_Universitatea_Bacau)))
```

The entity *student* now can be modeled thanks to reengineering in OWL [BAKU05], showing how to transform the description of an entity from the Common LISP dialect to OWL.

```
<owl:Class rdf:about="#student">
        <rdfs:subClassOf rdf:resource="#Faculty"/>
        <owl:onProperty rdf:resource="#Computer Science"/>
<owl:Class rdf:about="#Corp_C_Universitatea_Bacau"/>
        </owl:Class>
        <owl:ObjectProperty rdf:ID="Descriere_student">
          <rdfs:domain rdf:resource="#Student" />
          <rdfs:range  rdf:resource="#Media" />
        </owl:ObjectProperty>
      <owl:ObjectProperty rdf:ID="Bursier">
          <rdfs:subPropertyOf rdf:resource="#Are_Bursa" />
          <rdfs:range rdf:resource="#Bursa" />
        </owl:ObjectProperty>
</owl:Class>
```

The reason *web services* are different in comparison with other kind of web applications is because while HTML pages or web forms from ASP.Net web applications are dedicated to human final users, web services are to be understood by remote machines which refer them over the distributed system. Companies that are dealing with advertising over the Internet or E-commerce provide their own web services to other companies or to final users or just developers to access their information. The main idea is that web services have methods that access information from different data sources and serialize it in XML. The web methods are to be invoked remotely in the distributed system, network or Internet. Web services are the perfect solution for a web developer to build easily distributed applications, having in consideration the semantic prevalence, access data in short time and low costs, productiveness being in this case the suited word for describing this activity. As semantic web is, web services are platform independent, the client having the possibility to get and process XML data independently of his type, such as hardware platform, operating system, etc.

Grid computing may involve web services as well. In [WILEY04] grid computing is defined as connecting supercomputers into metacomputers that are remotely controlled. The greed reunites remotely placed computers for serving the scientific community in a more cohesive way. However, grid computing offers the possibility to have access to diverse resources which are not available from a single machine or a single kind of platform. Grid computing and *cluster computing* use distributed software reengineering in order to realize new objectives, to add new workstations or servers into the distributed system, to make possible cooperation and collaborative work within distributed systems so as to contain heterogeneous software platforms.

Languages such as Common LISP, OWL, HTML and XML as well as object oriented languages such as VisualC#.NET, Visual C++.NET or Java being integrated in distributed web applications, frameworks and developer kits in order to resolve problems of heterogeneity and distributiveness.

This paper's aim is not only to describe technologies, concepts already well known in the distributed application development area but also to reflect semantic evolution, metrics of semantic reengineering and the importance of software reengineering in the semantic approach. The next generation of web is for sure semantically oriented. The syntax is not the final objective, but just a way in achieving semantic efficiency and clarity of the ideas. Transformation and redefinition of data structures play an important role in this scenario as well as key factors of distributed applications reengineering.

3. Distributed Web Applications Semantic Reengineering Metrics

Reengineering becomes a must in distributed applications development, due to the efficiency it brings in the development cycle. Software companies or just research laboratories do not have the physical resources or time to start projects from scratch over and over again. Even if they had, it would have been inefficient and absolutely chaotic to abandon good work which has been already done with money and time spending and not to reuse it in the following projects. Experience and expertise have already been achieved by the specific situations in which the development team worked in the past in order to realize the objectives from the past. This expertise has to be implemented in the future projects, experience being important for them as well.

In [TOVA08] we described code transformation as being an important component of software reengineering while having in consideration aspects related to security, concurrency, openness, scalability, dependability or transparency. For distributed systems, as well as for distributed applications it is essential to follow this metrics as important approaches of software quality. This metrics must convey in which proportion, the software quality characteristics are realized by software developers. First of all, as a premise for reengineering the entity that would be subjected to it must have passed quality certification. Having passed quality certification time before guarantees a good premise for a successful reengineering process.

Concurrency is one of the most important distributed application's metrics. Concurrency is defined as a property of software processes to be executed in the same time, with the possibility that each process (thread) communicates with the others. In distributed applications, users may interact simultaneously with parts or domains of the application. When talking about distributed databases, data may be accessed from many physical locations. Software applications have to consent maximum concurrency with maximum safety. The increasing of concurrency should not affect the functionalities of the application. The database resources should be accessible to a big number of users that can be considered to be infinite in formalization.

$$\lim NrUsr(APP_i) = \infty \text{ ,where } NrUsr = \sum_i Usr_i \quad (3)$$

Shall we consider the rights each user has when accessing the software application, distinctions have to be made, depending of the category in which the user belongs. In this case, the formula has to be transformed in order to reflect the user's role about the application.

$$NrUsr = \sum_{i=1}^{n}\sum_{j=1}^{m} Usr_{ij}, \quad (4)$$

where the index *j* represents the category and *i* the index of the user from the category. *NrUsr* in this case have to be maximal and considered infinite as well.

While doing reengineering to the distributed application, each category of users should considered, and attention should be focused on the rights each category of users have. Optimization of user accounts is very important in distributed applications development. Categories and rights have a semantic background for the software developers. The associations between users and rights have semantic intelligence.

The ***openness*** characteristic is very common for distributed systems and it is also compulsory for a good and maintainable distributed application. Apart from autonomous

sequential software programs that cannot be modified nor may be configuration updatable during their compilation and execution, distributed systems and applications be updated during the execution nods or procedures being added, modified or deleted. A distributed application may change it's configuration during the lifecycle. The indicator ***Icop*** (openness indicator) reflects the number of actualization procedures during a period of time. It is advisable to compare these indicators from each two periods of time t_i, t_{i+1} and to build time series with them.

$$Icop = \sum_{i=1}^{n} Ins_i + \sum_{j=1}^{t} Upd_j + \sum_{k=1}^{r} Del_k \quad (5)$$

If it is desired to make comparisons between the numbers of actualizations from two periods of time it is very simple and efficient just forming a ratio index.

$$Incop = \frac{Icop_1}{Icop_0} \quad (6)$$

The openness of the application is in direct proportion with the ***Icop*** indicator. Every chain of transformations and modifications creates the reengineering of the distributed application. It is very important to have an open distributed application in order to make reengineering, but what is the most important factor in building reliable distributed applications is that the system works while transformations are being made.

Openness is important for the semantic transformation. An open application can be subjected to software reengineering easily by adding new components with new meanings and objectives.

Transparency and ***fault tolerance*** means that the user must not be restricted to use the application and the main functions of the system must not be affected if there are some components that do not work. Transparency means that from the user's point of view, the software application appears as a single non divided unit which helps him, without showing it's distributiveness. It is not important as well as not recommended to show the users the fragments or clusters of the software system. It will not be accessible to see the fragment or the cluster from which the information brought by the query resides, whether is ***replicated*** or subjected to ***concurrency*** while using it. Committing or finalizing transactions in distributed application is also done independently and transparently. In this case, reengineering may be more difficult in comparison with the traditional, non distributed applications because the development team must see the image behind the transparent image, as the back-side one.

The formalism that represents a grid from a distributed software application may be referred as :

$$Grid_i = \bigcup_{j=0}^{n} Frag_{ij} \quad (7)$$

as a reunion between the j indexed fragments of software or database fragments from the network. If we have replication in our distributed system, we must put one condition that will optimize the access to one replica and that the fragments should be distinct. We must evaluate each fragment or replica and see if there are any updates and modifications.

The whole image of the application will appear transparently and can be described by the following formalism as ***TrImage*** or transparent image.

$$Trimage = \bigcup_{i=0}^{m} \bigcup_{j=0}^{n} Frag_{ij} = \bigcup_{i=0}^{m} Grid_i \quad (8)$$

Transparency makes possible semantic constancy not depending of the deficient functionality of some modules of the system. Architectural reengineering may be also realized without notifying the users voluntarily or involuntarily by malfunctions or stops in the system's operation.

4. Conclusions

This paper described software reengineering in distributed applications on the subject of the hypertext entities semantic perspective. Semantic reengineering is very important and may be as well considered as the main component of software reengineering. Particular metrics have been shown regarding semantic reengineering and distributed applications. In addition to the text entities reengineering, development of distributed applications presumes technological implementations and platform dependence. However, the openness of distributed systems involves autonomy and context independence.

Semantic reengineering appears as an argument of the reengineering function, this concept being implemented and developed by the author.

As the other types of reengineering, semantic reengineering is also related with the renewing of the application's objectives, each objective being quantitatively measured as well as about the quality of the results.

References

1. Barzdins, G., Gruzitis, N. and Kudins, R. **Re-engineering OntoSem Ontology Towards OWL DL Compliance, Knowledge-Based Software Engineering,** in "Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering"
2. Eberhart, A. **Semantic Management of Distributed Web Applications,** IEEE DISTRIBUTED SYSTEMS ONLINE, 1541-4922, 2006, IEEE Computer Society Vol. 7, No. 5; May 2006
3. Ivan, I., Popa, M. and Tomozei, C. **Reingineria Entitatilor Text,** Revista Romana de Informatica si Automatica, vol. 15, no. 2, 2005
4. Newman, A. **A Relational View of the Semantic Web,** XML.COM, March 14, 2007
5. Plotikin, G. D. **A Structural Approach to Operational Semantics,** Computer Science Dept., Aarhus University, 1981
6. Tomozei, C. and Varlan, S. **Distributed Applications Reengineering Metrics,** Studii si Cercetari Stiintifice, no. 17, Seria Matematica, Universitatea din Bacau
7. Reilly, E. D. **Concise Encyclopedia of Computer Science,** Wiley, 2004
8. http://www.w3.org/TR/owl-features/ 07.03.2008
9. http://www.w3.org/2001/sw/ 07.03.2008
10. http://dsonline.computer.org/portal/site/dsonline/index.jsp

[1] Cosmin Tomozei is Universtiy Assistant at Mathematics and Computer Science Department from Faculty of Sciences of the University of Bacau. He PhD candidate from October 2007 at Economic Informatics Department from University of Economics, Bucharest. He holds an Master in Science - Databases- Business Support from University of Economics, Bucharest. He graduated in Economic Informatics at Faculty of Economic Cybernetics, Statistics and Informatics in 2006. His main research areas are: - object oriented programming; - functional programming; - software reengineering; - distributed applications.

[2] Codification of references:

[BAKU05]	Barzdins, G., Gruzitis, N. and Kudins, R. **Re-engineering OntoSem Ontology Towards OWL DL Compliance, Knowledge-Based Software Engineering,** in "Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering"
[EBER06]	Eberhart, A. **Semantic Management of Distributed Web Applications,** IEEE DISTRIBUTED SYSTEMS ONLINE, 1541-4922, 2006, IEEE Computer Society Vol. 7, No. 5; May 2006

[IVPOTO05]	Ivan, I., Popa, M. and Tomozei, C. **Reingineria Entitatilor Text,** Revista Romana de Informatica si Automatica, vol. 15, no. 2, 2005
[NEWM07]	Newman, A. **A Relational View of the Semantic Web,** XML.COM, March 14, 2007
[PLOTKIN81]	Plotikin, G. D. **A Structural Approach to Operational Semantics,** Computer Science Dept., Aarhus University, 1981
[TOVA08]	Tomozei, C. and Varlan, S. **Distributed Applications Reengineering Metrics,** Studii si Cercetari Stiintifice, no. 17, Seria Matematica, Universitatea din Bacau
[WILEY04]	Reilly, E. D. **Concise Encyclopedia of Computer Science,** Wiley, 2004
[www1]	http://www.w3.org/TR/owl-features/ 07.03.2008
[www2]	http://www.w3.org/2001/sw/ 07.03.2008
[www3]	http://dsonline.computer.org/portal/site/dsonline/index.jsp