A Dialog Based Game Component for a Competencies Based E-Learning Framework

Ciprian-Bogdan Chirila*
*University Politehnica of Timişoara, Romania
Faculty of Automation and Computer Science
E-mail: chirila@cs.upt.ro

Abstract—**Traditional education systems are highly theoretical and almost entirely based on student knowledge memorization. Nowadays european society development is focused on practical skills, globalization and competitiveness. Existing romanian e-learning systems tend to present and explain knowledge in a spectacular way using multimedia but none of them are based on the concept of competence. The solution in this sense is a new educational system which is based on competencies achieved in real life scenarios.**

I. Introduction

The current romanian educational system and some from other european countries tend to be highly based on information presentation, memorization and reproduction. Usually, in the classroom, the teacher presents the information on the white board expecting students to memorize it and then reproduce it during their exams. In some romanian schools e-learning systems [14] are used for this purpose. It is highly possible that students are unable to use the memorized information in their real life problems or in practical situations. Sometimes hypothetical situations are imagined by teachers in written or oral exam subjects which do not cope with real life problems.

We consider that the concept of competence is a quality participation of an individual in a real life situation. We consider competence also the capability of an individual to be able to use its acquired knowledge in order to solve or apply it in a practical problem or in order to overcome a certain situation [15].

In this paper we will present a dialog based game framework which allows teachers to create real life situations scenarios based on dialogs, images and student interaction in order to train student competencies for different disciplines. The framework is part of a complex e-learning system called "The Little Prince" inspired from the story with the same name [3] written by Antoine Saint-Exupery french writer, poet and pioneering aviator. We mention that our e-learning system has a front-end further referred as the *e-learning platform* and a back-end further referred as the *e-learning content manager*.

In figure 1 we can see the main blocks from of our competence based e-learning system: *i)* the dialog game file; *ii)* the e-learning management system; *iii)* the e-learning platform; *iv)* the database server; *v)* the dialog based game component.

The dialog game model is created and maintained by the teacher according to an imagined scenario and a set of rules

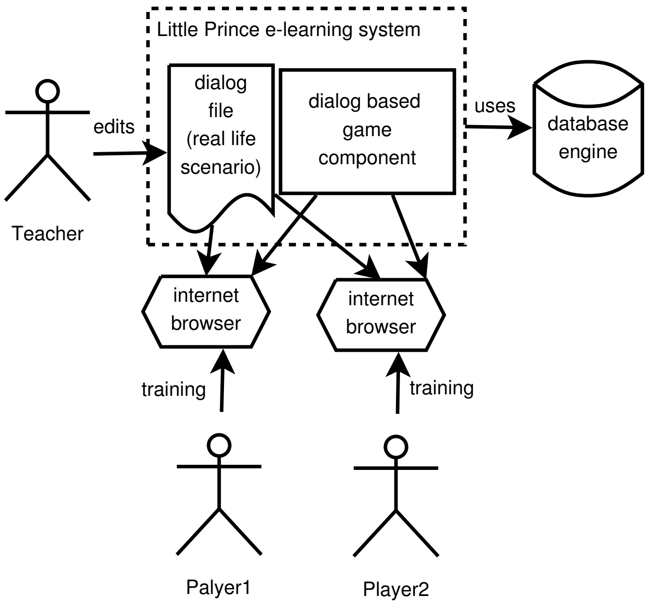

Fig. 1. The Dialog Based Game Framework

in order to train student competencies.

The e-learning management system is an online web application designed only for teachers in order to offer them support for loading and storing e-learning content, including the dialog game files.

The e-learning platform is an online web application designed for students where they can learn, train and evaluate themselves. The e-learning platform is a pluggable framework the dialog game based is a plugged in component.

Both web applications rely on a relational database engine through which they communicate indirectly.

The dialog game based component is plugged into the e-learning framework and thus it facilitates the interpretation of dialog game files in web browsers for the training students to play and train their competencies at the same time.

The paper is structured as follows. In section II we will present some rationale for the concept of competence and our competence based e-learning system. Section III presents our competence based e-learning model. Section IV describes the model of the dialog based game component. In section V we present a few details of the component implementation written in JavaScript. Section VI presents related works in the field of e-learning systems. Section VII concludes and sets the future work.

II. The Concept of Competence

In order to understand the concept of competence we must discuss the limits of the knowledge based learning. Nowadays in the classic learning system the student is overwhelmed with information in an unbalanced manner regarding his needs. Thus, the knowledge gets a viral behavior in the moment it is exploited at its real value.

Still some consider that the success of education is given by the number of individuals who serve the purpose of knowledge and not the mankind. The theories of knowledge are often insufficient in order to perform a practical task. So we can conclude that mankind is dominated by knowledge while it should be vice-versa knowledge must serve the mankind.

The usage of the concept of competence in education represents a revolution in this field. The purpose of education changes and refers the student quality participation in a supraindividual system. The competence gives us a power based also on the native skills of the student. Knowledge becomes a mean but not a goal and also a competence support.

The concept of competence takes into account this aspect. The competence based ideas are part of a social constructivism approach where the student is centered on both quality and quantity coordinates. The approach is based on student high attention and its authentic state, it opens the student for a correct and free thinking [7].

On the other hand if we consider the objectives of transmitting knowledge, the goal is set to how much the student knows and thus he becomes an encyclopedic, unadapted being. Such an approach is highly disrespectful for the student because the only important thing here is what the system wants and not what the student needs. The student is "doped" with someone else ideas hoping that someday they will be useful. The actual approach is based on transmitting information by the teacher behind its desk. This approach is based on the lower state based preoccupation. The student becomes the prisoner of the solutions that must be taken and reproduced.

III. The Competence Based e-Learning Model

The competence based e-learning model is based on a hierarchy of the competencies. In a top-down presentation on the first level we find the domains. The competence domains are the followings: *i)* communication in native language; *ii)* mathematical competencies; *iii)* communication in foreign languages; *iv)* scientific and technological competencies; *v)* learning how to learn competencies; *vi)* interpersonal relation and civic competencies; *vii)* initiative and entrepreneurial competencies; *viii)* cultural sensibility and artistic expression; *ix)* self-realization competencies. On the next level, competence domains are divided into general competencies. Further, general competencies are divided into specific competencies. Specific competencies are divided into competence variables. Finally, competence variables have actions attached grouped in lists. Action lists are grouped in activities, while activities are grouped in activity lists. Each action list is dedicated to one competence variable. Each activity from an activity list is dedicated to one specific competence.

IV. The Dialog Game Component Model

In this section we will present the core of the dialog based game framework model.

A. Rules and Structure

The dialog game is designed as an online conversation between two students. The dialog game is teacher driven because there are available only a few choices at each step for students to choose. The students will choose their sentences alternatively. One student can not choose two sentences in a row.

The dialog is a set of predefined sentences used in real life situations. The dialog game is written in XML format. In the next subsections we will explain into details each dialog game component. A dialog game contains: *i)* a role table; *ii)* a symbol table; *iii)* a verification condition table; *iv)* dialog sentences; *v)* semantical actions. In figure 2 we can see the skeleton of such an XML dialog file.

```
<dialog>
 <tabRoles>...</tabRoles>

 <tabSymbols>...</tabSymbols>

 <tabVariables nSpecificCompetenceId="...">
  ...
 </tabVariables>

 <tabVerificationConditions>
  ...
 </tabVerificationConditions>

 <tabSentences>...</tabSentences>
</dialog>
```

Fig. 2. Dialog Game XML Structure

B. Roles

A role is a set of behaviors, rights and obligation for an actor in a social situation. The student becomes an actor when he is playing the dialog game. Roles are stored in an XML list element that we consider the role table and is named <tabRoles>. Each role is defined in a <role> element. The "szGradeSymbol" attribute is used for storing the name of a symbol where the grades for that role is stored. We have to remember that each role is assigned to a student. In figure 3 we have an example of two roles: buyer and seller created for a shopping game scenario. In our released games we

```
<tabRoles>
 <role szGradeSymbol="nSellerMark">
  idSeller
 </role>
 <role szGradeSymbol="nBuyerMark">
  idBuyer
 </role>
</tabRoles>
```

Fig. 3. Role table

imagined several scenarios having roles like: seller and buyer in a growsery store, in an exchange office, in an electronics supermarket; mechanic and driver in a car service, mobile phone operator and customer in a mobile phone company, tailor and customer in a taylor shop etc.

C. Symbols

By symbols we denote JavaScript runtime variables active during the dialog game played online by two students. The values of the symbols may change during the game. The symbols were designed for: *i)* printing their values in dialog sentences; *ii)* reading a value from a dialog sentence and storing it into the symbol; *iii)* evaluating verification conditions in order to decide the correctness of the student statements. Verification conditions will be presented in details in subsection IV-E. Variables add an increased expression and realism to the dialog game. Since the games run in browsers on the client side, each student will have its own version of variables but the framework will synchronize them at different moments in order to store the same values and make the game consistent.

In figure 4 we present an example of a symbol table for a growsery shopping scenario in which we test the competence of addition and multiplication in a real life situation.

```
<tabSymbols>
 <symbol szName="nApplePrice" szType="int"
  nMinValue="1" nMaxValue="8">3</symbol>
 <symbol szName="nPearPrice" szType="int"
  nMinValue="2" nMaxValue="10">4</symbol>
 <symbol szName="nBananaPrice" szType="int"
  nMinValue="3" nMaxValue="12">5</symbol>

 <symbol szName="nAppleQuantity" szType="int"
  nMinValue="1" nMaxValue="4">0</symbol>
 <symbol szName="nPearQuantity" szType="int"
  nMinValue="1" nMaxValue="4">0</symbol>
 <symbol szName="nBananaQuantity" szType="int"
  nMinValue="1" nMaxValue="4">0</symbol>

 <symbol szName="nPrice" szType="int">
  10
 </symbol>
 <symbol szName="nAmount" szType="int">
  100
 </symbol>
 <symbol szName="nChange" szType="int">
  90
 </symbol>
</tabSymbols>
```

Fig. 4. Symbol table

Next, we will analyze the XML symbol element attributes.

The initial value of a symbol is stored into the body of the <symbol> XML element. The initial value is interpreted according to its declared type. For instance if the initial value is 4 and the variable type is integer then the value will be integer 4. If the variable type is string then the value will be the string ”4”. If the variable type is float or double then the value will be 4.0. We will get back to types later in this subsection.

A very important issue about initialization is that it can contain not only literal constants but also any arithmetical expressions in order to increase the expression power of the scenario. The goal is to generate random values for the symbols but still respecting a semantic of properties imposed by the scenario. The basic feature of the initialization is to generate integer of float numbers in a given range. Another feature of initialization is to control the number of decimals for the generated values. For example, the quantity of salt in a bread recipe must be in between certain limits in order to create a realistic scenario and not just a hypothetical one.

The name of the symbol is set as an XML attribute named ”szName”. The name of the symbol is a simple identifier beginning with a letter. For example the first symbol from the table in figure 4 is named ”nApplePrice”. An unwritten naming convention is used for the symbols: *i)* the first letter denotes the type (n for integer numbers, f for floating point numbers, sz for strings, tab for arrays); *ii)* the rest of the name is spelled in proper case.

The type of a symbol can be: *i)* integer; *ii)* float; *iii)* string; *iv)* array. Type declaration is mandatory since we store all symbols as strings and when verifications conditions are evaluated the symbol values must be converted to their typed values. For example if we want to check that the initial amount is equal with the payed amount plus the change we would write the following condition $v(”nAmount”) == v(”nPrice”) + v(”nChange”)$. Le us suppose that the value of ”nAmount” symbol is 10 and the value of ”nPrice” symbol is 7 and the ”nChange” is 3 then we need to operate with values of type integer where $10 = 7{+}3$. Not knowing the types of the symbols the comparison would operate with strings like in ””10” = ”7 + ”3”” where ”10” == ”37” which is false. This is because the additive operator in JavaScript is overloaded and because we must provide the correct typed values of the variables. To be noted that in order to access the value of a symbol one must use the v(”name”) function which has as argument the symbols name. The ”v” or value function is part of the dialog framework and it returns the correctly typed value according to its declared type.

The range of a symbol is set by two XML attributes named ”nMinValue” and ”nMaxValue”. Obviously they work only for numeric symbols and not for strings or array symbols. We designed these attributes for two reasons: *i)* in order to be able to perform range checking if the variables are written by a student and read at input by the framework in some sentence; *ii)* for generating random values in some points of the dialog game in order to simulate an autopilot behavior when the dialog is played by only one student and its discussion partner is the computer, namely the framework.

D. Competence Variables

Competence variables were described briefly in section III. This concept is also present in the context of the dialog game. Since a dialog game is built around the idea of training competencies we need to list and refer their identifiers for: *i)* attaching them the corresponding verification conditions when designing the dialog game; *ii)* reporting student performance grades to the e-learning platform at the end of the dialog game play. In figure 5 we listed the containing XML elements

```
<tabVariables>
 <variable nId="0438"
  nSpecificCompetenceId="227"
  szName="v0438"/>
 <variable nId="0439"
  nSpecificCompetenceId="227"
  szName="v0439"/>
</tabVariables>
```

Fig. 5. Competence variable table

<tabVariables> and <variable>. The former elements act as a container and the former element models the competence variable. A <variable> XML element has several attributes: *i)* "nId" is the integer value of the competence variable identifier from the e-learning system database; *ii)* "nSpecificCompetence" is the integer value of the specific competence identifier from the same database; *iii)* "szName" is a freely chosen identifier at game design time to be referred by verification conditions.

E. Verification Conditions

Verification conditions are the hearth of the competence evaluation. They are boolean expressions respecting the syntax of JavaScript language and containing value accesses to symbols. Verification conditions are more suitable to check competences from mathematics discipline than other disciplines. For instance in order to check the addition and multiplication competence one could design a dialog in a growsery store where the buyer asks for a few kg of apples knowing their price, hands out an amount to the seller and the seller must give him the correct change. Such a verification condition can be written with the symbols declared in symbol table listed in figure 4:

$$v("nAmount") ==$$
$$v("nApplePrice")*v("nAppleQuantity")+v("nChange")$$

This verification condition can be evaluated on the seller side because he is the one writing the amount, but also on the buyers side if he gives a correct or wrong change. Verification conditions are listed in a table like in figure 6. In figure 6 we listed four verification conditions embedded in the XML element named <tabVerificationConditions>: *i)* the first one checks if the price computed for the bought fruits is correct; *ii)* the second checks if the same price is computed incorrectly; *iii)* the third verifies if the given change is correctly computed; *iv)* the fourth verifies if the given change is incorrectly computed. Actually, we have two verification conditions each one has a negated version. The motivation for this duplication is that in some sentences a correct answer may involve recognizing partner mistakes. In such a situation the very same but negated verification condition must be evaluated. One can notice that verification conditions are expressed through XML elements named <verificationCondition>. The first attribute for a verification condition element is the "name" which denotes identifiers for later references. Verification conditions will be referred from dialog sentences and the evaluation will be detailed in subsection IV-F. The second attribute is named "szVariableName" and its value points to a variables table

```
<tabVerificationConditions>
 <verificationCondition name="vc1"
  szVariableName="v0439">
  v("nApplePrice")*v("nAppleQuantity")+
  v("nPearPrice")*v("nPearQuantity")+
  v("nBananaPrice")*v("nBananaQuantity")
  ==v("nPrice")
 </verificationCondition>

 <verificationCondition name="vc1n"
  szVariableName="v0439">
  v("nApplePrice")*v("nAppleQuantity")+
  v("nPearPrice")*v("nPearQuantity")+
  v("nBananaPrice")*v("nBananaQuantity")
  !=v("nPrice")
 </verificationCondition>

 <verificationCondition name="vc2"
  szVariableName="v0439">
  v("nAmount")==v("nPrice")+v("nChange")
 </verificationCondition>

 <verificationCondition name="vc2n"
  szVariableName="v0439">
  v("nAmount")!=v("nPrice")+v("nChange")
 </verificationCondition>
</tabVerificationConditions>
```

Fig. 6. Verification condition table

entry. This referring mechanism is used to link the competence variable with the corresponding verification conditions.

F. Dialog Sentences and Variants

A sentence represents a grammatical unit formed out of words which have a semantic relation with the words that precede and follow it. The sentence is said by a student playing a role in order to participate in a discussion. Sentences are modeled by <sentence> XML elements which are grouped in a table named <tabSentences>. Sentences are formed of <variant> XML elements which represent several variants of the same dialog semantics in order to be chosen at one point in the discussion. The reason for having several variants in the same sentence are: *i)* to allow selecting the variant that suits better the role player state of mind or its nature, for example, one player which is more polite will choose the variant from the current sentence which expresses better this feature; *ii)* to be able to choose a variant for changing the flow of the dialog according to the choice made, for example, in a discussion at some point one player may choose to buy apples and another one to buy pears; *iii)* to be able to choose a correct variant in a given context, which will be used for evaluation. For example, a player who buys some fruits may have to choose from two different variants expressing the correct and incorrect change he has to receive. If he choses the right variant he will get a good grade for the trained competence, otherwise he will get a poor grade. If we look at the sentences from the technical point of view we can notice that they are equipped with attributes like: *i)* "nState" - identifying the state in which the sentence is played; *ii)* "nRoleIdentifier" - referring the role to which the sentence belongs to; *iii)* "szImage" - referring a relative image path to an image to be shown according to the context of the sentence. The dialog is modeled as a state machine where each sentence represents a state. The dialog starts at state 0 and each variant drives a transition into a different state. The last state of the automaton is set to be -1. When the game component reaches the -1 state the game play is stopped. Each sentence is designed to be played by a role player and the next sentence will be player by the other role player. The image is set in order to offer a visual representation of the current state of the dialog according to past made choices. In figure 7 we present an example of a sentence corresponding to the first state in automaton, state 0. The sentence belongs to the buyer role and the sentence refers one image which depicts a buyer entering a growsery store and engaging in a conversation with the seller.

```
<tabSentences>
 ...
 <sentence nState="0" szRoleId="idBuyer"
  szImage="/img/01.jpg">
  <variant id="41" nNextState="5">
   Hello, how much cost the fruits ?
  </variant>
<variant id="42" nNextState="5">
   Hi, how much for the fruits ?
  </variant>
 </sentence>
 ...
</tabSentences>
```

Fig. 7. Sentence table

G. *Values and Fields*

In order to favor diversity and to create interesting and diverse dialog sentences and in order to get feedback from the players we use in the sentence texts special XML elements named values and fields. A <value> XML element will be used to display the value of a symbol in the text of the sentence. Thus, using symbols with random generated values one dialog game played several times will always behave differently and the correct answers can not be memorized and reused illegally. A <field> XML element can be used to generate a text field in the sentence so the student can enter data when giving a reply in a dialog play. Fields are linked to symbols together with their types and range such that an invalid input will be detected, warned and not validated.

H. *Semantical Actions*

Semantical actions represent JavaScript code which is to be run or evaluated and they are of three types: *i)* <semanticAction> - is used when a new state is entered and it is located inside a sentence; *ii)* <preSemanticAction> - is run before the evaluation of a verification condition and it is located inside a variant; *iii)* <postSemanticAction> - is run after the evaluation of a verification condition and it is located inside a variant. The main use for such semantical actions is to set symbols to different computed values necessary in the current context of the dialog. This usage is typical to state semantical actions.

Pre-semantic actions are designed for reusing symbols and preparing them with special values for the very next verification condition.

Post-semnatic actions are designed for cleaning, reseting symbols with specific values needed in the next states of the dialog.

Semantical actions of any type can be also used for debug purposes like printing in the browser console the value of some symbols.

V. Prototype Implementation

The code of the prototype is written in JavaScript [4] and is using the jQuery [11] library for HTML element manipulation.

The code was tested on browsers like: Mozilla Firefox^{TM}[6], Google Chrome^{TM}[9], Apple Safari^{TM}[8], Opera^{TM}[1]. From the technical point of view the dialog game component: *i)* manipulates div HTML elements; *ii)* fills them with texts and images; *iii)* evaluates conditions and executes semantical actions; *iv)* transfers texts and the symbol table from one browser to another through a server-side chat protocol.

The chat protocol consists in a few calls: *i)*startGame(object) is called by the server side when a new dialog game is initiated; *ii)* stopGame(object) is called by the client in order to notify the server side that the dialog game is over and reports the player results; *iii)* sendMessage(object) is called at each step of the game when the game control is handled from one player to the other. The structure of the parameter object is well structured. The object passed when starting the game contains the player identifier, name and role identifier. The object passed to the server side and the end of the game contains the mark achieved by the player. The object passed with the "sendMessage" call contains the table of symbols and the currently chosen sentence.

The integration of the dialog game component consists in creating a zipped file with all the dependencies and uploading it into the e-learning system. Such uploads should occur rarely only when an upgrade is made to the dialog game component.

The integration of a dialog game file into the e-learning system consists is uploading a zipped archive containing the dialog XML file and the referred images in a predefined directory structure.

For testing the dialog game component we used the FireBug [5] plugin for Mozilla Firefox^{TM}where each step of the game an all variables can be inspected. In order to favor more the testing process we created an offline chat protocol for loading both player game windows into the very same browser as frames. The dialog game developer can test its game locally without accessing the e-learning framework.

The most occurring errors were due to floating point arithmetical expressions.

VI. Related Works

In [2] is presented an e-learning system embedding semantic web technologies. One of the components are able to evaluate open questions by semantic similarity measurements of the student answer versus teacher answer. In our work the evaluation is done automatically but not using open questions. The presented e-learning system results accuracy is based on parameters while ours is exact, based on mathematical expressions evaluation. AeL [13] is a romanian learning management system. The AeL e-learning system has media rich online lessons which are based on curricula and practically on knowledge while the Little Prince e-learning system has not the same extent but is oriented on competencies. In [10] is presented an engine for producing e-learning content based on semantic metadata. Our work is similar in this sense the engine is the dialog component and the metadata is the dialog file. In cite [12] are presented challenges regarding the semantic web, social software and technology agents in e-learning environments. Our work includes learning semantics, the framework is a social software and the verification conditions related code from the dialog component behave like a software agent.

VII. Conclusions and Future Work

We conclude that we build a role playing game component for an e-learning framework. Students participate as players in driven dialogs this developing their competencies. Teachers can express their dialog scenarios by writing simple XML elements. The XML elements are filled with texts and images and they are grouped as a directed graph of states. XML elements embed mathematical expressions to be checked at game play in order to assess players competencies.

A JSON implementation for the dialog file keeping the same structure would eliminate the XML parsing stage, but its syntax seems to be complex to the teachers writing the scenarios.

The symbols used in the dialog texts as field or value elements increase the degree of diversity and variability of the scenarios. Thus, the same scenario can be played several times with different values for symbols creating different situations and demanding different answers.

The authors of scenarios may need a minimum knowledge of programming skills but they do not have to be professional software developers. Without such a dialog based game component the scenario authors would have to write the same JavaScript code from the component for each independent game they would like to develop. Such a task can be performed only by professional programmers only. Thus, the dialog based game component offers the possibility to non-programmers with a small programming knowledge to develop their own dialog scenarios.

Using the dialog based game component the author can focus on the scenario rather than on implementation details like: HTML elements manipulation, dialog state transitions, animations, chat protocols etc.

The XML structure contained in the game file offers a certain semantics to the dialog game. In some situations new additional semantics may be necessary and for that purpose we designed the semantical actions at the level of symbol initialization, sentence, pre- and post- verification conditions. We can not say that it can cover any imagined semantics but it can help to achieve it.

In the case of a dialog game play where one partner is lost because of some connection errors or when a human partner is not available online for a game play we can consider building a software agent capable of playing instead of a human. For such an agent to be consistent with the dialog scenario the game file should contain formulas for computing the correct answers not just formulas for checking the correctness of the given answers. The agent can be configured to give correct or bad answers deliberately in order to stimulate its human partner.

Acknowledgments

We would like to thank to all members of the Little Prince project www.miculprint.eu developed under the POSDRU contract number 85/1.1/S/64320 and namely to: Ioan Vlaşin the general manager of the project, Andraş Sorin the software team manager, Iulian Necea the head of the e-learning system development team, Samuel Bogdan the graphic interface designer.

References

[1] Opera Software ASA. Opera. https://www.opera.com, 2013.
[2] Dagoberto Castellanos-Nieves, Jesualdo Toms Fernndez-Breis, Rafael Valencia-Garca, and Rodrigo Martnez-Bjar. A semantic web technologies-based system for students assessment in e-learning environments. In *IADIS International Conference e-Learning 2007*, 2007.
[3] Antoine de Saint-Exupery. The little prince, 1943.
[4] Mozilla Foundation. Java script. https://www.mozilla.org, 2011.
[5] Mozilla Foundation. Firebug. https://www.mozilla.org, 2013.
[6] Mozilla Foundation. Mozilla firefox. https://www.mozilla.org, 2013.
[7] Timothy W. Gallwey. *The Inner Game of Work*. Random House, New York, 2000.
[8] Apple Inc. Safari. https://www.apple.com/safari, 2013.
[9] Google Inc. Google chrome. https://www.google.com/chrome, 2013.
[10] A.Seyedeh Sara Mousavi Jabbari and B.Seyed Ehsan Mousavi Jabbari. Semantic metadata in an engine producing e-learning content. *International Journal of e-Education, e-Business, e-Management and e-Learning*, 1(4), October 2011.
[11] The jQuery Foundation. jquery project. https://www.jquery.com, 2013.
[12] Bolanle A. Olaniran. Challenges facing the semantic web and social software as communication technology agents in e-learning environments. *18 International Journal of Virtual and Personal Learning Environments*, 1(4), October-December 2010.
[13] Siveco Romania. Advanced e-learning. http://www.advancedelearning.com/.
[14] Marc J. Rosenberg. *e-Learning: Strategies for Delivering Knowledge in the Digital Age*. McGraw-Hill, 2001.
[15] Ioan Vlaşin. *Competenţa - O participare de calitate*. Editura Unirea, Alba Iulia, Romania, 2013.